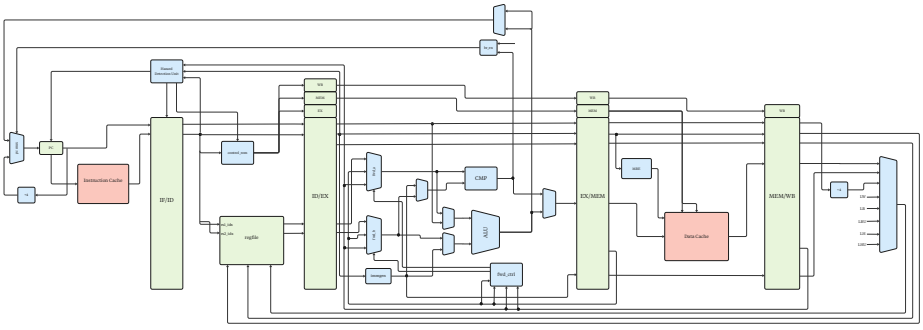
bitbangers CPU

Devul, Sanjana, Aditya

Overview of Design

- `RV32IMA_Zicsr_Zifencei` (with some `sv32` support).
- 5-stage pipelined in-order, static not-taken branch prediction, etc., etc.
- CLINT (interrupt controller compatible w/privileged spec)
- Behavioral UART for co-simulation.

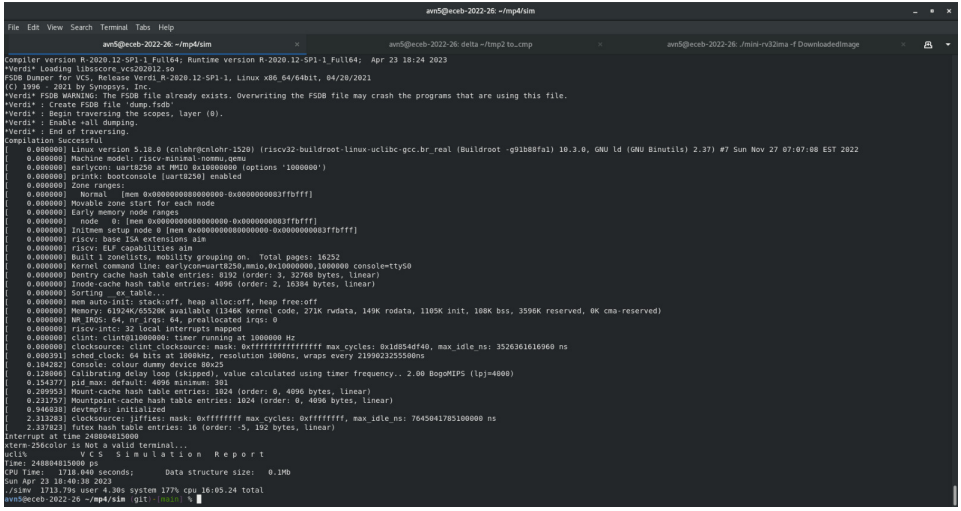
Datapath


Instruction Cache
IF/ID
ID/EX
EX/MEM
MEM/WB
Data Cache
regfile
CMP
ALU
Fwd_ctrl

More on features

- Atomics: LR/SC are implemented without reservation stations since it's a uniprocessor.
- AMOs (read-modify-write) are implemented by trapping and emulating using LR/SC.
- Zicsr — difficult extension to do correctly since lots of edge cases. Chose to stall for four cycles to ensure no hazards, since enforcing privileged spec's read/write permission constraints plus hazards was very expensive in area and did not improve performance since CSR instructions are rare.
- Zifenci — implementing simple write-update snooping-based cache coherence between I$ and D$ to prevent flushes of D$. (Bus contention is not a concern since I$ is never written to.) This way, D$ never has to write back every cacheline on `fence.i`, the instruction and data streams are always in sync via snooping.

Booting nommu Linux…to some point.



Why is nommu Nontrivial?

- Requires building Linux in a certain way:
  - uClibc instead of glibc, make sure the busybox initramfs is OK w/this.
  - Create a flat binary instead of ELF so that we don't need a bootloader/firmware.
  - Requires writing a DTB (device tree blob) and loading into memory.
- Testing this in simulation is…slow.
- Booting even nommu Linux creates very large log files — our `dump.fsdb` touches 3GiB+ on the previous boot.[1]

[1] EWS has a 10GiB/user ceiling that's easy to hit :/

What's the Solution?

- Write a cycle-accurate model of the processor, interrupt controller, other peripherals in C.
- Boot Linux there, then use the software emulator as a specification for what the hardware should implement.
- This allows us to have a golden model to compare instruction traces and processor state with.
- Debugging the software emulator is much easier, since it's about 400 lines of C (instead of ~3k lines of Verilog) and boots Linux in ~5 seconds.
- Hardware/software co-design is a very useful tool, and likely the only way to actually do this project.

Software emulator

- We (very heavily) modified an RV32 emulator to be cycle-accurate with our processor.
- Additional difficulties: once interrupts are turned on, instruction traces can differ a lot due to emulator and simulated timers operating at different frequencies.
- Needed to make the emulator cycle-accurate down to timing of interrupts between instructions.

Writing a correct emulator that can boot Linux is hard!

- Has involved building Linux many times in lots of different (experimental) ways.
- Building the firmware (OpenSBI) with various configurations (RISC-V boot process is still a WIP and changes often).
- Lots of reading Linux/OpenSBI source.
- When comparing traces w/QEMU, finding abnormalities in how QEMU handles DTB — needed to read QEMU source and build QEMU with changes.
- ...and of course, reading the RISC-V specs.

Some results

- Simulation ~1 million instructions takes 1 minute on VCS.
- Emulator takes ~5 seconds to boot Linux (which is about 137 million instructions till the `login` program.)
- Synthesis successful:
  - Slack: 1.41
  - Area: 18889.7237 $\approx$ 18900

Edge cases!

- Linux finds edge cases w/the processor very easily.
- Correctness is extremely important.
- Finding bugs is hard, since Linux assumes the underlying hardware is perfect, and fails silently — finding a failure in execution means comparing traces w/emulator.

Examples of bugs

- A load to precisely `x2` followed by `mret` caused a hazard to be incorrectly detected, resulting in the `mret` never happening:

```
ld x2, (xX)
mret
```

  The symptom was an access to uninitialized memory — figuring out the issue was with return-from-trap required reworking the emulator's timers till the traces matched exactly.
- Interrupt flag was losing state: this was happening when an interrupt happened precisely *after* a multiplication operation — the stall in EX caused certain side effects.
- An interrupt occurs at exactly the instruction used to turn interrupts off.
- ...so many more

Currently working on

- Further debugging `nommu` to boot.
- Testing the new MMU/ `sv32` in the emulator and booting MMU Linux in emulation.
  - Caveat: flattening MMU Linux doesn't seem possible, so we need to use firmware to boot an ELF Linux — currently exploring OpenSBI's `fw_payload` and comparing emulator traces w/QEMU.
- Finishing `sv32` in hardware and (finally) booting Linux there.

What we should've done differently

- Started earlier — this project is probably a 2.5 month project, not a 1 month one.
- Tried booting a smaller, simpler RISC-V operating system — unfortunately these are rare.
- Started FPGA prototyping earlier — waiting for Linux to boot in simulation makes TDD *very* slow.

Q/A