

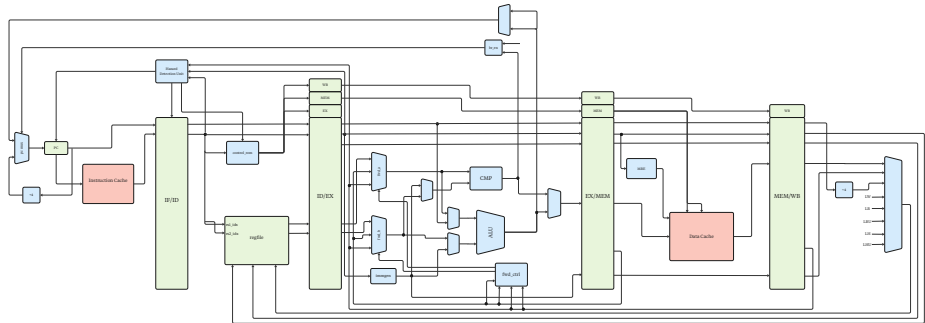
bitbangers CPU

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



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.

```
avn5@eceb-2022-26: ~/mp4/sim
File Edit View Search Terminal Tabs Help
avn5@eceb-2022-26: ~/mp4/sim
Compiler version R-2020.12-SP1-1 Full64; Runtime version R-2020.12-SP1-1 Full64; Apr 23 18:24 2023
*Verdi*: Loading Libscore vcs202012.5e
FSDB Dumper for VCS, Release Verdi R-2020.12-SP1-1, Linux x86_64/64bit, 04/20/2021
(C) 1996 - 2021 by Synopsys, Inc.
*Verdi*: FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi*: Create FSDB file 'dump.fsdb'.
*Verdi*: Begin traversing the scopes, layer (0).
*Verdi*: Enable +all dumping.
*Verdi*: End of traversing.
Compilation Successful
[ 0.000000] Linux version 5.10.0 (cnlohr@cnlohr-1520) (riscv32-buildroot-linux-uclibc-gcc-br_real (Buildroot -g91b88fal) 10.3.0, GNU ld (GNU Binutils) 2.37) #7 Sun Nov 27 07:07:08 EST 2022
[ 0.000000] Machine model: riscv-minimal-nommu,qemu
[ 0.000000] earlycon: uart8250 at MMIO 0x10000000 (options '1000000')
[ 0.000000] printk: bootconsole [uart8250] enabled
[ 0.000000] Zone ranges:
[ 0.000000]   Normal [mem 0x0000000000000000-0x0000000003ffbfff]
[ 0.000000] Movable zone start for each node
[ 0.000000] Early memory node ranges
[ 0.000000]   node 0: [mem 0x0000000000000000-0x0000000003ffbfff]
[ 0.000000] Initmem setup node 0 [mem 0x0000000000000000-0x0000000003ffbfff]
[ 0.000000] riscv: base ISA extensions ain
[ 0.000000] riscv: ELF capabilities ain
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 16252
[ 0.000000] Kernel command line: earlycon=uart8250,mmio,0x10000000,1000000 console=ttyS0
[ 0.000000] Dentry cache hash table entries: 8192 (order: 3, 32768 bytes, linear)
[ 0.000000] Inode-cache hash table entries: 4096 (order: 2, 16384 bytes, linear)
[ 0.000000] Sorting __ex table...
[ 0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[ 0.000000] Memory: 6192K/65520K available (1346K kernel code, 271K rdata, 149K rodata, 1105K init, 108K bss, 3596K reserved, 0K cma-reserved)
[ 0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[ 0.000000] riscv-intc: 32 local interrupts mapped
[ 0.000000] clint: clint@11000000: timer running at 1000000 Hz
[ 0.000000] clocksource: clint_clocksource: mask: 0xffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616960 ns
[ 0.000391] sched clock: 64 bits at 1000kHz, resolution 1000ns, wraps every 2199023255900ns
[ 0.104282] Console: colour dummy device 80x25
[ 0.128006] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=4000)
[ 0.154377] pid_max: default: 4096 minimum: 301
[ 0.209953] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.231757] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.940020] devtmpfs: initialized
[ 2.312283] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 7645041705100000 ns
[ 2.337023] futex hash table entries: 10 (order: -5, 192 bytes, linear)
Interrupt at time 248004015000
xterm-256color is Not a valid terminal...
ucli VCS Simulation Report
Time: 248004015000 ps
CPU time: 1719.040 seconds; Data structure size: 0.1Mb
Sun Apr 23 18:40:38 2023
./simv 1713.79s user 4.30s system 177% cpu 16:05.24 total
avn5@eceb-2022-26: ~/mp4/sim git:[main] %
```

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

¹EWS 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