RISC-V Architectural Conformance of the Vreteno Core

Filip Filmar

***Abstract*—Vreteno is a three-stage RV32IMAC core with machine, supervisor and user modes and Sv32 paging, written in the hardware description language TxHDL. This report runs the official RISC-V architectural tests on it. The tests come from RISC-V International's ACT4 suite, and the Sail reference model computes the result each test must reproduce. Each test runs on the core's Verilog netlist under Verilator. Of the 1028 tests in the catalog, 156 apply to Vreteno, in 22 suites. 156 of them pass on the netlist, and 156 pass their own self-check on Sail. Together the tests compare 28 308 values with Sail's. The first run, on 2026-10-07, had one failure, and it is a real defect: the core cannot set `mstatus.TVM`. A second finding of that run is that the core has no `menvcfg`, so it conforms to version 1.11 of the privileged specification rather than 1.12. Both were filed against TxHDL and fixed by 2026-10-08. The run of that day found a new defect: the core's new data cache returned a stale word to a load right after a store to the same line. It was filed, and fixed by 2026-10-09. The build that produces this report fetches every tool by checksum, and it runs every day against the latest commit of the core.**

***Index Terms*—RISC-V, architectural tests, conformance, Sail, Verilator, hardware verification, TxHDL.**

Contents

I Introduction 1
I-A How to read this report . . . 1
II The architectural tests 1
II-A Self-checking tests . . . 2
II-B Which tests apply . . . 2
III The device under test 2
III-A The wrapper . . . 2
III-B Booting into the test . . . 2
III-C The memory and the end of a run . . 2
IV Describing Vreteno to the tests 2
IV-A Extensions and parameters . . . 3
IV-B What is left out . . . 3
V The build 3
V-A From test file to target . . . 3
V-B Known failures . . . 3
V-C Every day . . . 3
VI Results 3
VI-A Cycles . . . 3
VII Findings 4
VII-A `mstatus.TVM`, `TW` and `TSR` cannot be set 4
VII-B No `menvcfg` and no `senvcfg` . . . 4
VII-C Since the first run . . . 4
VII-D A choice that is not a defect . . . 4
VIII What this run does not check 4
IX Conclusion 5
Appendix A: The failing tests 5
Appendix B: Every test 5
References 5

The author is independent, at f@hdlfactory.com.

This report is generated. A build of the repository `filmil/vreteno-conformance` runs the tests and typesets the report from the results of that run, every day. Every count, table and chart in it comes from that build.

The author used a large language model, Claude, as an assistant in building the test harness and in writing this report; every commit of the repository says so and lists the prompts that led to it, verbatim.

Built by Bazel from commit `f572c8da82e5` of `vreteno-conformance`, with TxHDL commit `cacb24130d7a`, on 2026-11-05.

I. Introduction

A processor conforms to RISC-V when every instruction it implements gives the result the specification defines. RISC-V International publishes a suite of tests for this, the architectural tests [1]. Each test runs a fixed program and checks the values it computes against values the Sail reference model [3] computed for the same program. A core that passes a test agrees with the reference model on everything that test checks.

Vreteno is the RISC-V core of the TxHDL project [5]. It implements RV32IMAC: the base integer set, multiply and divide, the atomic instructions and the compressed ones. It also has machine, supervisor and user modes, Sv32 virtual memory, and an instruction cache. TxHDL writes the core as a Rust program and lowers it to Verilog. The project checks that netlist against its own reference model in lockstep, and it runs on an Artix-7 board.

Those checks say that the core does what its authors meant. The architectural tests say something else: that what the authors meant is what the specification says. This report applies them.

The result is short. Of the 1028 tests in the catalog, 156 apply to Vreteno, and 156 pass on its netlist, which is 100.0 %. The tests compare 28 308 values with Sail's. Every defect the runs have found in the core so far is described in Section VII, with the issue that tracks it.

A. How to read this report

Section II says how the tests work. Section III describes the device under test: the netlist, the memory around it, and how a run ends. Section IV says how Vreteno is described to the tests, and why each choice was made. Section V describes the build that runs it all. Sections VI and VII give the results and the defects found. Section VIII lists what this run does not check. The appendices give the console output of each failing test and a line for every test.

II. The architectural tests

The architectural tests are assembly programs, one file each, grouped into suites by extension. This report uses ACT4, the current framework of the suite, at commit `a215271eab12` [1], [2]. Each test file starts with a header that states what the test needs: the extensions it requires, its `-march` string, and any parameter of the core it depends on, such as XLEN or the number of physical memory protection entries.

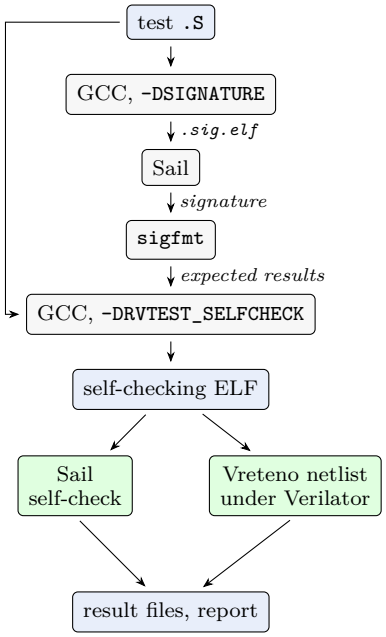

Fig. 1. How one test becomes two results. Every box is a Bazel action.

A. Self-checking tests

An ACT4 test checks itself. It is built twice, and the reference model runs in between. Fig. 1 shows the four steps.

1) The test is compiled with `-DSIGNATURE`. Built this way, it stores every result it computes into a region of memory called the signature.
2) Sail runs that program and writes the signature region to a file.
3) The signature is turned into assembler source, one `.word` directive per value.
4) The test is compiled again with `-DRVTEST_SELFCHECK`, including that source. Now each time the test computes a value, it compares it with Sail's value at the same place, and branches to a failure routine on a mismatch.

The final program decides for itself whether it passed. It prints `TEST PASSED` or `TEST FAILED` on a console, and on a failure it also prints the instruction, the register, the expected value and the value it got. A device under test only has to run the program, provide the console, and report how the program ended.

B. Which tests apply

A test applies to a core when the core implements every extension the test requires and has every parameter value the test states. The ACT4 framework makes this selection in Python, from a description of the core in the RISC-V Unified Database format. This build makes the same selection in Bazel's own language, from a list of extensions and parameters (Section IV). It applies the same rules, including comparisons such as `NUM_PMP_ENTRIES: '>0'`.

The selection is made over 1028 tests: the RV32 suites and the privileged suites of the catalog. It keeps 156 of them.

III. The device under test

The device under test is Vreteno's netlist, the Verilog that TxHDL writes for the core. It is the same netlist that the board's bitstream is built from, so the tests check what is synthesized rather than a model of it.

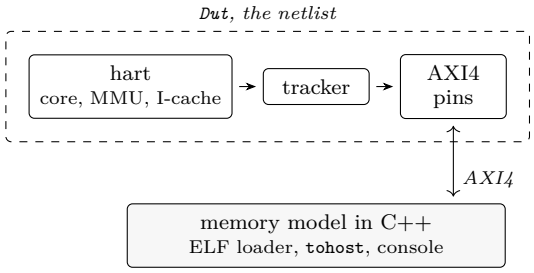

Fig. 2. The device under test. Only the memory model is written for this testbench; everything inside the dashed box is TxHDL's.

A. The wrapper

The core on its own speaks TxHDL's transaction channels, not a standard bus. A small TxHDL unit, `Dut`, joins three existing parts so that the netlist has one standard AXI4 manager port (Fig. 2). The three parts are the hart, which is the core with its memory management unit; the tracker, which turns the core's loads, stores and fetches into AXI4 bursts; and an adapter that puts the five AXI4 channels on pins. All three come from TxHDL unchanged. The wrapper is about 250 lines of Rust, most of them port names, and TxHDL lowers it with the core into one Verilog file.

B. Booting into the test

Vreteno fetches its first 4 KiB from a memory inside the core, which the netlist initializes. The wrapper puts two instructions there: `lui t0,0x40000` and `jr t0`. They jump to `0x4000_0000`, where every test is linked. On the board, that address is the start of DDR3, and the core's instruction cache holds words from that range. So the tests run from cached memory, as the board's programs do.

C. The memory and the end of a run

The testbench is a C++ program around the Verilated netlist. It loads the test's ELF into a sparse memory and answers every AXI4 burst from it, with every `ready` held high. The interrupt lines and the debug module's lines are held low.

A test ends the way it ends on Sail, through the HTIF `tohost` word. The testbench reads the word's address from the ELF's symbol table. A write to its upper half completes a command. Device 1 with command 1 prints a character on the console. Any other command with an odd lower half ends the run: 1 means the test passed, and 3 means it failed.

The testbench writes a result file for each run: the status, the cycles, the instructions retired, and the console text. A run that reaches 20 million cycles stops with the status `TIMEOUT`. The build does not stop when a test fails. The result file records the failure, a test target reads it, and this report counts it.

IV. Describing Vreteno to the tests

The tests need to know what the core implements. Three files tell them, and a fourth tells the reference model. All four describe the same machine, and they are written from the core's source in TxHDL, `cpu/vreteno/src/isa.rs` and `core.rs`.

- `rvtest_config.h` defines a macro for each extension and parameter. ACT4 generates this file from a Unified Database description, with Ruby tools; here it is written by hand.

TABLE I
CHOICES IN THE DESCRIPTION OF VRETENO

Choice	Reason in the core
Privileged version 1.11	No `menvcfg` or `senvcfg` (Section VII).
`mtvec`, `stvec` direct only	The core clears the two mode bits on a write.
Misaligned access traps	A load or store that is not aligned raises an address-misaligned exception.
Misaligned AMO traps as misaligned	The specification allows this or an access fault; the core does the former.
`mtval` nonzero	The core writes the instruction for an illegal instruction, the pc for a breakpoint, the address for a misaligned or faulting access.
A and D bits: Svade	The page walker never writes; a clear A bit, or a clear D bit on a store, is a page fault.
No PMP, no `mcountinhibit`	Neither is implemented.
`mvendorid` and the rest 0	All four read as zero.

- `rvmodel_macros.h` says how the device ends a test and prints a character. Vreteno's testbench speaks HTIF as Sail does, so one set of macros serves both.
- `link.ld` places the test at `0x4000_0000`.
- `sail.json` configures Sail to behave as Vreteno does where the specification lets an implementation choose.

A. Extensions and parameters

The extensions are I, M, Zmmul, A, Zaamo, Zalrsc, C, Zca, Zicsr, Zifencei, Zicntr, Zihintpause, Zihintntl, Sm, S, U, Sv32, Svbare, Svade. The parameters are XLEN of 32, no misaligned loads and stores in hardware, and no physical memory protection. Table I gives the choices that matter for the tests and the reason for each.

B. What is left out

One suite is left out by hand. The Svade tests begin by clearing `menvcfgh.ADUE`, a register that exists only from privileged version 1.12. On a 1.11 core the first instruction of the test traps, and Sail also fails the test in that configuration. The behaviour Svade names, a page fault on a clear A or D bit, is still tested by the Sv suite.

Everything else the selection drops, it drops for a stated reason: the test needs an extension the core lacks, such as F, D or the bit manipulation extensions, or it needs a parameter the core does not have, such as physical memory protection entries or misaligned access in hardware.

V. THE BUILD

Bazel runs everything in this report, and it fetches every tool it uses by checksum or by commit. Nothing is installed on the machine beyond Bazelisk and Git. Table II lists the tools.

A. From test file to target

A repository rule downloads the test suite and reads the configuration header of every test file. It writes the headers into a list that the build reads when it analyses the targets, so that selecting the tests needs no Python. For each selected test, a Bazel macro makes one target that builds the self-checking ELF in the four steps of Fig. 1. It makes two more targets that run the ELF, one on Sail and one on the netlist, and a test target for each run. `sigfmt`, the tool in step three, is a short Rust program. It replaces the Python function in ACT4 that does the same job.

TABLE II
WHAT THE BUILD FETCHES

Tool	Version and source
Vreteno	TxHDL commit `cacb24130d7a`, from its GitHub mirror
Tests	riscv-arch-test commit `a215271eab12`
Reference	Sail RISC-V model 0.15, prebuilt release
Compiler	riscv-none-elf-gcc (xPack GNU RISC-V Embedded GCC x86_64) 15.2.0
Simulator	Verilator 5.046, TxHDL's prebuilt binary
C and C++	LLVM over a pinned Debian sysroot, from TxHDL
LaTeX	a pinned TinyTeX, with TxHDL's copies of IEEEtran, pgf and listings

Each step is an action with declared inputs and outputs, so Bazel runs the 156 tests in parallel and caches every step. A change to the core reruns the netlist runs and nothing else. A change to the description of the core rebuilds every ELF.

B. Known failures

A test that the core is known to fail is listed with the URL of the issue that tracks it. Its test target then passes while the run fails, and fails once the run passes. The list therefore shrinks when the core is fixed, and `bazel test` stays useful for changes to the harness while a defect is open. This report reads the result files, not the test targets, so a known failure counts as a failure here.

C. Every day

A workflow runs the build every day against the newest commit of TxHDL. It typesets this report, publishes it as a release of the repository, and publishes the same results as a web page on hdlfactory.com. The commit of the core and the date are in the title footnote of every copy.

VI. RESULTS

On 2026-11-05, 156 of the 156 tests passed on Vreteno's netlist. Of the tests that failed there, 0 are known failures, each tracked by an issue, and 0 are new. 156 of the 156 tests passed their self-check on Sail. The Sail column is a check on the harness rather than on the core: each self-checking program must pass on the model that wrote its expected values. When it does, the description of the core, the compiler flags and the test sources agree with one another.

Table III gives the results by suite. The checks column counts the values a test compares with Sail's. The counts exclude the fill of the unused signature area and the marker words. The suite that compares the most is `I`, with 10 176 values in 39 tests.

A. Cycles

The 156 runs took 3 540 311 cycles to retire 627 654 instructions, 5.64 cycles per instruction on average. Fig. 3 breaks this down by suite.

These numbers describe the testbench as much as the core. An architectural test is long straight-line code that runs once. The instruction cache then helps only within a line of four

TABLE III
RESULTS BY SUITE

Suite	Tests	Sail	Netlist	Checks	CPI
ExceptionsSv	2	2	2	108	4.67
ExceptionsSvSm	4	4	4	920	3.49
ExceptionsSvZaamo	2	2	2	44	4.93
ExceptionsSvZaamoSm	3	3	3	262	3.47
ExceptionsSvZalrsc	2	2	2	76	4.70
ExceptionsSvZalrscSm	3	3	3	518	3.38
I	39	39	39	10 176	6.46
M	8	8	8	3 184	8.10
Sv	25	25	25	1 016	4.91
SvSm	6	6	6	88	5.50
Svbare	2	2	2	20	6.00
SvbareSm	1	1	1	14	6.12
Zaamo	9	9	9	4 890	6.37
Zalrsc	2	2	2	648	4.48
Zca	26	26	26	3 600	6.19
Zicntr	2	2	2	20	6.03
Zicsr	6	6	6	1 068	6.75
Zifencei	1	1	1	10	8.86
Zihintntl	4	4	4	24	5.94
ZihintntlZca	4	4	4	24	5.82
Zihintpause	1	1	1	6	5.93
Zmmul	4	4	4	1 592	6.83
Total	156	156	156	28 308	5.64

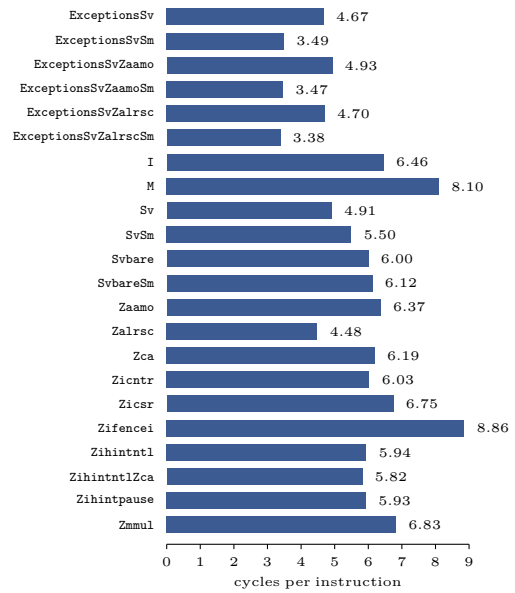

Fig. 3. Mean cycles per instruction on the netlist, by suite.

words, and each new line is fetched over the bus. The slowest suite is `Zifencei`, at 8.86 cycles per instruction, and the fastest is `ExceptionsSvZalrscSm`, at 3.38. The multiply and divide instructions take several cycles each in Vreteno's sequencer, which is why `M` and `Zmmul` are above the rest. The figures are a record that the runs completed at a plausible speed. They are not a performance claim.

VII. Findings

The first run, on 2026-10-07 against TxHDL commit `894cc2e3b296`, found two departures from the specification. Both are filed as issues against TxHDL, and both are in the privileged architecture. In that run, every unprivileged instruction the core implements matched Sail on every value the tests check. A later run that fails anything else shows it in Section VI and in Appendix A, as a failure that is not known.

A. *`mstatus.TVM`, `TW` and `TSR` cannot be set*

The test `SvSm_sv_mstatus_tvm_test-00` sets `mstatus.MPP` to 3 and `mstatus.TVM` to 1, then reads `mstatus` back. Sail reads `0x00101800`. Vreteno reads `0x00001800`: the TVM bit, bit 20, did not stick. The test's own console output is in Appendix A.

The cause is in the core's write mask for `mstatus`, `MSTATUS_W = 0x000e_19aa`, which leaves out bits 20, 21 and 22: TVM, TW and TSR. The privileged specification requires all three to be writable when supervisor mode is implemented [4]. Without them, machine mode cannot trap a supervisor's `satp` accesses and `sfence.vma`, nor its `wfi` and `sret`. A hypervisor or a monitor that emulates those instructions relies on these traps. The issue is TxHDL issue 1347 [7].

B. *No `menvcfg` and no `senvcfg`*

Version 1.12 of the privileged specification adds `menvcfg`, which a core must have when it implements user mode, and `senvcfg`, which it must have when it implements supervisor mode. Vreteno has neither, and an access to either raises an illegal instruction exception. Described as a 1.12 core, Vreteno fails every test: the tests write `menvcfg` while they boot. Described as a 1.11 core, it passes all but one.

The core does have `mstatush`, which is also new in 1.12, so it sits between the two versions. This report describes it as 1.11, and the Svade suite, which needs `menvcfgh`, does not run. The issue is TxHDL issue 1348 [8]. Adding the two registers, with their fields read-only zero where the core lacks the feature, would let the core be described as 1.12 and the Svade suite run.

C. *Since the first run*

TxHDL fixed both issues by commit `48daf9778add`, and the run of 2026-10-08 against that commit passes `SvSm_sv_mstatus_tvm_test-00`. The core now has `menvcfg` and `senvcfg` too, but this report still describes it as a 1.11 core, so the Svade suite still does not run.

The same commit adds a data cache for the DDR3, and with it five tests fail that passed before: the byte, halfword and word loads, `I-lb-00`, `I-lbu-00`, `I-lh-00`, `I-lhu-00` and `I-lw-00`. Each fails where it stores a new value to a line the cache holds and loads it back at once. The load returns the value the case before it stored. The tests run in the cached range of the address space, and the uncached path has no such failure. The issue is TxHDL issue 1431 [9]. TxHDL fixed it by commit `e9681300a8b2`, and the run of 2026-10-09 against that commit passes all five tests.

D. *A choice that is not a defect*

The first run had six more failures, all in tests of atomic instructions at misaligned addresses. Vreteno raised address-misaligned there, and the first configuration of Sail expected an access fault. The specification allows either. The fix was in `sail.json`, which now says that the core raises address-misaligned, and the six tests pass.

VIII. What this run does not check

A passing run says that Vreteno agrees with Sail on the values the selected tests check. It does not say more than that, and five gaps are worth stating.

- **Interrupts.** The testbench raises no interrupt, so the tests that need a timer or an interrupt controller are not

selected. The core's interrupt path is checked by TxHDL's own tests, not here.

- **The Svade suite** does not run, for the reason in Section VII.
- **One simulator.** The netlist runs under Verilator only. TxHDL also writes VHDL, and its own tests run that under nvc, but the architectural tests do not.
- **The description is written by hand.** If `rvtest_config.h` or `sail.json` claims a behaviour the core does not have, the tests check the claim. A self-check on Sail catches a description that disagrees with itself, not one that disagrees with the core. The table of choices in Section IV is what a reviewer should check against the core's source.
- **The board.** The tests run on the netlist in simulation, with a memory model, not on the Artix-7 board with its DDR3.

IX. Conclusion

The official RISC-V architectural tests now run on Vreteno's netlist, every day, with every tool fetched by the build. In this run, 156 of the 156 tests that apply to the core pass. The first run found a real defect: `mstatus.TVM`, `TW` and `TSR` cannot be set. The run also found that the core lacks the environment configuration registers of privileged version 1.12.

Both were filed, and TxHDL fixed both the next day. The run that day showed the loop working: the TVM test started to pass, its entry in the list of known failures had to go, and the same run caught a new defect in the core's data cache within hours of its commit. That defect was fixed the day after, and the five tests it broke pass again.

Appendix A
The failing tests

No test failed on the netlist in this run.

Appendix B
Every test

Tables IV and after list each test with the values it checks, the cycles its run took on the netlist, and its cycles per instruction. A test's name is shown without the name of its suite.

References

[1] RISC-V International, "RISC-V Architectural Certification Tests," https://github.com/riscv-non-isa/riscv-arch-test, commit `a215271eab12`.

[2] RISC-V International, "Certification Test Plan," https://riscv.github.io/riscv-arch-test/ctp.html.

[3] RISC-V International, "Sail RISC-V model," https://github.com/riscv/sail-riscv, release 0.15.

[4] RISC-V International, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*, versions 1.11 and 1.12.

[5] F. Filmar and D. Janković, "TxHDL: A Hardware Description Language That Is a Rust Library," https://www.hdlfactory.com/txhdl/, 2026.

[6] W. Snyder et al., "Verilator," https://verilator.org.

[7] TxHDL issue 1347, "mstatus.TVM, TW and TSR are not writable," https://git.hdlfactory.com/HDL/txhdl/issues/1347.

[8] TxHDL issue 1348, "no menvcfg or senvcfg, so the core is privileged version 1.11," https://git.hdlfactory.com/HDL/txhdl/issues/1348.

[9] TxHDL issue 1431, "a load right after a store to a cached line returns the line's old word," https://git.hdlfactory.com/HDL/txhdl/issues/1431.

TABLE IV
Every test on the netlist (1 of 4)

Test	Checks	Cycles	CPI	Result
ExceptionsSv				
sv32_exceptions_Smode-00	54	15 070	4.85	pass
sv32_exceptions_Umode-00	54	17 661	4.53	pass
ExceptionsSvSm				
sv32_exceptions_mprv_S_Mmode-00	222	36 932	3.90	pass
sv32_exceptions_mprv_U_Mmode-00	222	35 303	3.83	pass
sv32_medeleg_Smode-00	238	49 655	3.23	pass
sv32_medeleg_Umode-00	238	49 561	3.30	pass
ExceptionsSvZaamo				
sv32_exceptions_Zaamo_Smode-00	22	11 940	5.26	pass
sv32_exceptions_Zaamo_Umode-00	22	14 393	4.70	pass
ExceptionsSvZaamoSm				
sv32_exceptions_Zaamo_mprv_S_Mmode-00	98	24 331	4.11	pass
sv32_medeleg_Zaamo_Smode-00	82	36 793	3.25	pass
sv32_medeleg_Zaamo_Umode-00	82	36 699	3.34	pass
ExceptionsSvZalrsc				
sv32_exceptions_Zalrsc_Smode-00	38	13 625	4.93	pass
sv32_exceptions_Zalrsc_Umode-00	38	16 144	4.53	pass
ExceptionsSvZalrscSm				
sv32_exceptions_Zalrsc_mprv_S_Mmode-00	194	32 894	3.75	pass
sv32_medeleg_Zalrsc_Smode-00	162	43 781	3.23	pass
sv32_medeleg_Zalrsc_Umode-00	162	43 687	3.31	pass
I				
add-00	398	30 548	6.80	pass
addi-00	374	24 647	6.94	pass
and-00	398	30 603	6.79	pass
andi-00	374	24 654	6.95	pass
auipc-00	62	10 174	6.47	pass
beq-00	270	32 093	6.63	pass
bge-00	270	32 872	6.76	pass
bgeu-00	270	32 857	6.75	pass
blt-00	270	32 974	6.77	pass
bltu-00	270	32 851	6.73	pass
bne-00	270	33 784	6.89	pass
fence-00	6	9 432	5.88	pass
jal-00	114	78 872	4.54	pass
jalr-00	242	25 911	6.04	pass
lb-00	130	14 411	6.53	pass
lbu-00	130	14 418	6.51	pass
lh-00	126	13 971	6.58	pass
lhu-00	126	13 994	6.57	pass
lui-00	62	10 137	6.52	pass
lw-00	126	13 763	6.62	pass
nop-00	6	8 158	6.24	pass
or-00	398	30 560	6.80	pass
ori-00	374	24 634	6.96	pass
sb-00	210	15 467	6.30	pass
sh-00	202	15 100	6.28	pass
sll-00	398	30 545	6.80	pass
slli-00	202	17 079	6.75	pass
slt-00	394	30 339	6.80	pass

TABLE V
Every test on the netlist (2 of 4)

Test	Checks	Cycles	CPI	Result
I				
slti-00	370	24 575	6.94	pass
sltiu-00	370	24 598	6.93	pass
sltu-00	394	30 381	6.78	pass
sra-00	398	30 534	6.80	pass
srai-00	202	17 077	6.75	pass
srl-00	398	30 582	6.80	pass
srli-00	202	17 073	6.75	pass
sub-00	398	30 576	6.79	pass
sw-00	200	14 935	6.26	pass
xor-00	398	30 562	6.79	pass
xori-00	374	24 652	6.95	pass
M				
div-00	398	42 043	9.34	pass
divu-00	398	42 100	9.34	pass
mul-00	398	30 998	6.88	pass
mulh-00	398	30 991	6.87	pass
mulhsu-00	398	31 072	6.85	pass
mulhu-00	398	31 016	6.86	pass
rem-00	398	42 099	9.36	pass
remu-00	398	42 074	9.35	pass
Sv				
sv32_VA_all_ones_Smode-00	6	9 182	5.86	pass
sv32_VA_all_zeros_Smode-00	6	9 202	5.83	pass
sv32_global_pte_Smode-00	18	9 789	6.00	pass
sv32_global_pte_Umode-00	18	12 332	5.11	pass
sv32_invalid_pte_Smode-00	34	12 715	5.19	pass
sv32_invalid_pte_Umode-00	34	14 035	4.93	pass
sv32_misaligned_page_Smode-00	18	10 850	5.52	pass
sv32_misaligned_page_Umode-00	18	11 640	5.37	pass
sv32_mstatus_mxr_Smode-00	42	13 759	5.21	pass
sv32_mstatus_mxr_Umode-00	42	16 179	4.73	pass
sv32_nleaf_pte_DAU_Smode-00	5	8 163	6.03	pass
sv32_nleaf_pte_DAU_Umode-00	5	8 163	6.03	pass
sv32_nleaf_pte_level0_Smode-00	18	10 977	5.51	pass
sv32_nleaf_pte_level0_Umode-00	18	11 744	5.36	pass
sv32_pte_reserved_rwx_Smode-00	6	8 223	5.96	pass
sv32_pte_reserved_rwx_Umode-00	6	8 223	5.96	pass
sv32_pte_rsw_Smode-00	30	11 520	6.01	pass
sv32_pte_rsw_Umode-00	30	15 187	4.89	pass
sv32_satp_access_Smode-00	30	11 395	5.44	pass
sv32_spage_Smode-00	82	18 870	4.88	pass
sv32_spage_access_Umode-00	154	33 299	3.89	pass
sv32_spage_mstatus_sum_set_Smode-00	54	15 621	4.96	pass
sv32_upage_Umode-00	82	24 655	4.21	pass
sv32_upage_mstatus_sum_set_Smode-00	106	21 910	4.53	pass
sv32_upage_mstatus_sum_unset_Smode-00	154	27 711	4.15	pass

TABLE VI
EVERY TEST ON THE NETLIST (3 OF 4)

Test	Checks	Cycles	CPI	Result
SvSm				
sv32_mstatus_mprv_Smode-00	10	9 069	5.91	pass
sv32_mstatus_mprv_Umode-00	10	9 026	5.91	pass
sv32_satp_access_Mmode-00	6	7 803	6.17	pass
sv32_upage_mprv_set_sum_set_Smode-00	10	9 313	5.70	pass
sv32_upage_mprv_set_sum_unset_Smode-00	26	11 429	4.96	pass
sv_mstatus_tvm_test-00	26	10 267	4.90	pass
Svbare				
Svbare_Smode-00	10	8 023	6.12	pass
Svbare_Umode-00	10	8 708	5.90	pass
SvbareSm				
Svbare_mstatus_mprv-00	14	8 206	6.12	pass
Zaamo				
amoadd.w-00	490	52 101	6.35	pass
amoand.w-00	490	52 104	6.35	pass
amomax.w-00	610	62 315	6.38	pass
amomaxu.w-00	610	62 335	6.38	pass
amomin.w-00	610	62 321	6.38	pass
amominu.w-00	610	62 334	6.38	pass
amoor.w-00	490	52 087	6.36	pass
amoswap.w-00	490	52 126	6.35	pass
amoxor.w-00	490	52 111	6.35	pass
Zalrsc				
lr.w-00	106	13 057	6.56	pass
sc.w-00	542	119 209	4.33	pass
Zca				
c.add-00	298	23 778	6.41	pass
c.addi-00	222	17 540	6.56	pass
c.addi16sp-00	66	11 209	6.34	pass
c.addi4spn-00	266	20 339	6.85	pass
c.and-00	198	18 374	6.38	pass
c.andi-00	182	15 746	6.54	pass
c.beqz-00	42	10 295	6.31	pass
c.bnez-00	42	10 295	6.38	pass
c.j-00	46	24 199	4.99	pass
c.jal-00	46	24 239	4.98	pass
c.jalr-00	78	12 815	6.21	pass
c.jr-00	46	12 514	5.87	pass
c.li-00	114	13 069	6.56	pass
c.lui-00	110	12 992	6.56	pass
c.lw-00	66	10 827	6.36	pass
c.lwsp-00	106	12 787	6.19	pass
c.mv-00	142	15 479	6.31	pass
c.nop-00	6	8 527	6.06	pass
c.or-00	198	18 354	6.39	pass
c.slli-00	210	17 549	6.77	pass
c.srai-00	186	16 048	6.55	pass
c.srli-00	186	16 048	6.55	pass
c.sub-00	198	18 374	6.38	pass
c.sw-00	126	12 046	6.04	pass

TABLE VII
EVERY TEST ON THE NETLIST (4 OF 4)

Test	Checks	Cycles	CPI	Result
Zca				
c.swsp-00	222	14 459	6.19	pass
c.xor-00	198	18 374	6.38	pass
Zicntr				
csrrc-00	10	8 734	6.03	pass
csrrs-00	10	8 734	6.03	pass
Zicsr				
csrrc-00	222	17 869	6.77	pass
csrrci-00	134	12 952	6.71	pass
csrrs-00	222	17 834	6.77	pass
csrrsi-00	134	12 953	6.71	pass
csrrw-00	222	17 840	6.78	pass
csrrwi-00	134	12 948	6.71	pass
Zifencei				
fence.i-00	10	13 249	8.86	pass
Zihintntl				
ntl.all-00	6	8 398	5.93	pass
ntl.p1-00	6	8 378	5.95	pass
ntl.pall-00	6	8 418	5.91	pass
ntl.s1-00	6	8 378	5.95	pass
ZihintntlZca				
c.ntl.all-00	6	8 508	5.82	pass
c.ntl.p1-00	6	8 488	5.84	pass
c.ntl.pall-00	6	8 528	5.80	pass
c.ntl.s1-00	6	8 488	5.84	pass
Zihintpause				
pause-00	6	8 398	5.93	pass
Zmmul				
mul-00	398	31 078	6.84	pass
mulh-00	398	31 071	6.84	pass
mulhsu-00	398	31 152	6.82	pass
mulhu-00	398	31 096	6.83	pass