

UNIVERSITY OF CALIFORNIA, BERKELEY

BERKELEY • DAVIS • IRVINE • LOS ANGELES • MERCED • RIVERSIDE • SAN DIEGO • SAN FRANCISCO • SANTA BARBARA •
SANTA CRUZ

Professor Koushik Sen
Computer Science Division, 735 Soda Hall #1776
University of California, Berkeley CA 94720-1776
PHONE: (510) 642-2420 EMAIL: ksen@cs.berkeley.edu

September 14, 2026

To whom it may concern:

It is my great honor to write this letter of recommendation for Kevin Laeuer, my Ph.D. advisee at Berkeley from 2017 to 2024 and now a Research Associate at Cornell. Kevin is super smart, hard-working, and driven, with excellent communication skills, and he is an exemplary mentor. He has the full potential to become a world-class researcher and teacher.

Kevin joined UC Berkeley in Fall 2016 as an exchange student from Germany and took my graduate course on Advanced Topics in Testing, Debugging, and Program Analysis. Impressed by his drive, dedication, and attention to detail, I encouraged him to apply for a Ph.D. He was accepted at all top schools, including UC Berkeley, MIT, and Stanford; I was thrilled to keep him at Berkeley.

Kevin's research spans hardware design verification, testing, debugging, and repair. His early work brought coverage-guided fuzzing (CGF) — the algorithm behind tools such as Google's AFL that has found vast numbers of critical bugs and security vulnerabilities in real-world software — to RTL circuits, where decades of attempts to automate stimuli generation from coverage feedback had failed to gain adoption. Kevin proposed an approach that requires minimal setup and exploits FPGA-accelerated simulation for rapid testing. He found that resetting a circuit is a crucial bottleneck and implemented transformation passes that make it feasible to reset arbitrary RTL designs quickly, a requirement for deterministic test execution. He released his implementation, RFUZZ, as open-source software. Jonathan Bachrach and I were hands-off; Kevin figured it all out himself. The paper appeared at ICCAD 2018, one of the two top conferences in computer-aided design — rare for a first-year Ph.D. student working with so little help from his advisors — and has inspired several subsequent research efforts on hardware fuzzing.

While working on RFUZZ, Kevin realized that fuzzing could improve with richer coverage information, which most open-source and academic RTL simulators do not support. His key insight: if every simulator implements a single new cover primitive, then line, toggle, finite-state-machine, and other metrics can be implemented as compiler passes, independent of the backend simulator. He built a prototype for the Chisel hardware construction language, demonstrated it across three software simulators, the FPGA-accelerated FireSim simulator, and a formal tool, and showed for the first time that detailed line coverage can be collected while booting Linux on FireSim at 65 MHz. This ground-breaking work, published at ASPLOS 2023 and released publicly, lets any simulator obtain rich coverage metrics for free. Again, Kevin ran the show: he conceived, implemented, and validated the idea; I merely listened in our one-on-one meetings.

Kevin also made a major contribution to the widely-popular open-source Chisel ecosystem by adding bounded model checking to the ChiselTest library, making a formal technique accessible to regular developers. His tool showed how hardware generators make even simple formal techniques more powerful, enabling checkers parameterized on signal sizes, FIFO depths, and more, with exact replay so dynamic-testing debug infrastructure can be reused. The upstream Chisel project used it to verify library

components; it has uncovered several bugs in FIFO implementations, inspired further community work, and been integrated into a formal verification course at UC Santa Cruz.

Toward the end of his Ph.D., Kevin reframed automated program repair as a debugging tool that provides rapid repair suggestions. His repair tool for Verilog RTL circuits combines symbolic repair templates with SMT-based bounded model checking, and scales to long failing tests via an adaptable unrolling window around the bug manifestation time step. It correctly repairs half of the benchmarks, versus one-third for the state-of-the-art tool CirFix, and answers in under ten seconds where CirFix often takes minutes or hours. The paper, RTL-Repair, appeared at ASPLOS 2024, alongside a second ASPLOS 2024 paper, Zoomie, on software-style debugging for FPGAs, which Kevin co-authored with Tianrui Wei.

Kevin defended his dissertation, *Automated Testing, Verification and Repair of RTL Hardware Designs*, in 2024 and joined Cornell as a Research Associate working with Adrian Sampson. I have followed his work closely since, and I want to describe what he has done there, because it shows him operating as an independent researcher rather than as a student. His work has moved from finding bugs toward what happens after a bug is found: the debugging and specification tools that hardware engineers actually spend their days in.

The first example is Surfer, an open-source waveform viewer published at CAV 2025. A waveform viewer is what an engineer opens when a simulation or a model checker produces a failing trace, and for many years the only full-featured open-source option was GTKWave, whose aging code base has proved very hard to extend. Surfer is written in Rust and was started at Linköping University; Kevin's part is the waveform backend, a library called wellen that he wrote and maintains under his own GitHub account. It parses the file header first so the signal hierarchy shows up almost immediately and the tool never needs a loading screen. For VCD files it keeps every signal in a compressed in-memory form, borrowing ideas from the FST file format, and decompresses only the signals the user selects, which also lets it parse a single VCD file in parallel chunks. On a 2.9 GiB VCD file, GTKWave needs 16 seconds and 800 MiB of memory; Surfer loads it in 2 seconds on a four-core machine and holds the signals in 74 MiB. The same compressed representation powers a server mode: a trace that lives on a compute farm can be opened remotely, and the local viewer pulls down only the signals the user selects, so nobody has to copy a multi-gigabyte VCD file to a laptop first. Because the parser uses only the safe subset of Rust, Surfer can be embedded in web services that open files uploaded by strangers. Wellen has since been adopted outside Surfer, by the VaporView plugin for Visual Studio Code and by Antmicro's trace2power tool, and the CAV artifact evaluation committee gave Surfer both its Available and Reusable badges.

Surfer goes well beyond fast file loading. It decodes values of typed signals from languages such as Chisel and Spade instead of showing raw bit vectors, ships RISC-V instruction decoders, compiles to WebAssembly so it runs in a browser, exposes a remote-control protocol modeled on the Language Server Protocol, and is the first open-source viewer to support the CXXRTL protocol for interactive simulation. It is embedded in TinyTapeout's browser-based chip design flow and in the quicksilicon and MakerChip teaching platforms, two commercial verification companies have built it into prototypes of their products, and courses at Cornell, MIT, UC Santa Cruz, and JKU Linz point their students to it. The project is now stewarded by the FOSSi Foundation and has NLnet funding for its next round of features. My favorite detail from the paper is that a master's student who wanted a waveform viewer that understands Chisel types, a thesis that would have been hopeless from scratch, could build it as an extension of Surfer. I mention all of this because Surfer is a tool that people use, which is a different kind of contribution from Kevin's Berkeley papers and one that few academics manage.

The second example is Paso, a Cornell project that Kevin leads with Adrian Sampson; the paper is currently under submission. To test a hardware module one writes two programs: a driver that turns high-

level transactions, such as a bus read, into cycle-by-cycle signal values, and a monitor that watches the signals and recovers the transactions. Both encode the same protocol, and writing them separately is a standing invitation for the two to disagree. Recent work from Adrian's group and others addresses this with type systems that encode protocols, but those require the design to be rewritten in a new language and cover only particular families of interfaces. Paso takes the opposite position: give up static guarantees, work on existing synchronous RTL without rewriting it, and make testing and debugging better. In Paso a protocol is written once, as a short imperative program that assigns input ports, advances the clock, and asserts on outputs; a fork statement says where the next transaction may overlap this one, which is how a pipelined interface is described. The same program runs forward, as a driver inside an interpreter, and backward, as a monitor. The backward direction is the technical heart of the paper: a reconstructor that takes a protocol and a waveform and infers the sequence of transactions that explains the waveform, or reports a protocol violation if none exists. The reconstructor is inspired by dynamic symbolic execution, the technique behind my own DART work, and I was pleased to see Kevin put it to a new use. Kevin and his students designed the language so that this actually works: loop guards may mention only design ports and constants, transaction arguments are immutable, every constraint reduces to a simple equality or inequality with a constant, and the clock step acts as a barrier, so the tool streams a trace cycle by cycle with no backtracking. Paso also has carefully worked-out rules that keep a testbench from observing transient values and that make concurrent transactions race-free; both are common causes of the flaky SystemVerilog testbenches that hardware teams know too well.

Kevin and his students wrote Paso specifications for twelve open-source designs, including an AES core, APB and AXI-Stream components, a Wishbone SRAM and FIFO, and register files from two RISC-V cores, plus a seventeen-protocol specification of the Wishbone bus covering all of its burst modes. The reconstructor processes a 31-million-cycle trace from an Ethernet MAC in about eleven minutes. More to the point, it found a real bug in the wrapped-burst address calculation of Antmicro's Wishbone SRAM, code that had already been merged into the LiteX SoC framework, and it flagged all five protocol-violation bugs from Ma et al.'s published dataset of FPGA bugs while accepting the fixed versions of the same designs. The first three authors are Cornell students: Ernest Ng, a Ph.D. student Kevin co-advises, and Nikil Shyamsunder and Francis Pham, who joined the project as undergraduates and wrote an earlier FMCAD 2025 student forum paper on the interpreter.

The interpreter side of Paso matters on its own. No open-source simulator can run UVM testbenches, and a recent study found that 88 percent of the fifty most popular FPGA projects on GitHub include no executable testbench code; which suggests that people most likely test by staring at waveforms instead. Paso gives such projects executable protocol checks that run on open-source simulators. The model of the design that Paso uses is deliberately general enough to sit on top of software simulation, FPGA emulation, or a formal engine, and Kevin describes his current focus as specification languages that serve testing and formal verification at the same time. Given that he has already built the fuzzer, the coverage infrastructure, the model checker, and the repair tool for RTL, he is unusually well placed to do it.

Kevin's proposal-writing skills are remarkable. He conceived the ChiselTest formal verification project and wrote a full proposal for a highly competitive Semiconductor Research Corporation (SRC) grant; Jonathan Bachrach and I only helped structure it. It was funded for three years, rare for a second-year graduate student. He repeated this at Cornell: the Paso project is funded by a \$156,400 Jane Street Hardware Research Collaboration Grant awarded in 2026, which Kevin wrote with Adrian Sampson as PI. He will have no problem raising funds. His open-source participation is equally exemplary: an active member of the Chisel community since 2019, maintainer of the official ChiselTest verification library since 2020, and maintainer of wellen.

Teaching is Kevin's passion, and he does it really well. He taught three distinct classes at UC Berkeley as a TA and guest lecturer, and from 2022 through 2025 he gave the formal verification lecture each year in Scott Beamer's agile hardware design course at UC Santa Cruz, using interactive Jupyter notebooks built on the verification support he added to Chisel. In Fall 2025 he was a Visiting Lecturer for CS 3410, Cornell's computer systems course, co-teaching with Giulia Guidi. He rebuilt the lab sections around worksheets and student collaboration; the labs had been one of the least popular parts of the course, and the feedback afterwards was very positive. His philosophy centers on feedback-driven instructional design, respecting students' time, and using experiments and visualization; he is keen to develop a class on automated bug-finding and verification.

Kevin has mentored over ten undergraduate and graduate students with a student-centered approach, and at Cornell this has grown into what is effectively a small research group. He informally co-advises two Cornell Ph.D. students with Adrian Sampson and one Princeton Ph.D. student with Mae Milano, has supervised two M.Eng. students, one of whom is now a Ph.D. student at Brown, and works with several undergraduates, one of whom is now at NVIDIA. He has also mentored two students through SIGPLAN's long-term mentoring program since 2021. One former mentee wrote: "He is an exceptional mentor and I have no doubt that Kevin will improve the lives of many students to come." Another: "It would not be an overstatement to say that the Ph.D. position I got today owes partly to Kevin."

Kevin also carries his share of service. He reviews for IEEE TCAD, was an external reviewer for ASPLOS 2025, and has served on the program committee of the LATTE workshop, the DDVC special session at DSD 2023, and the PLDI 2020 artifact evaluation committee. At Berkeley he co-organized visit days for over a hundred admitted students, which the department recognized with the 2020 EECS Chairs' Graduate Award, and as a student reviewer on the Ph.D. admissions committee he made a point of flagging candidates from non-traditional backgrounds so that at least one professor would read their files.

Kevin is a complete package with the DNA of a research and teaching leader: visionary, independent, driven, hard-working, an excellent communicator, and a perfectionist. In two years at Cornell he has written a funded grant, published a tool that other projects have adopted, and brought three students to a conference submission of their own; he is already doing the job of a faculty member. He is a better version of Michael Pradel, my former post-doc and now a full professor at the University of Stuttgart, Germany. I give Kevin my strongest recommendation without any reservations for a faculty position in your department.

Sincerely,

Koushik Sen
Professor, Computer Science Division
University of California, Berkeley