Noetic

Custom state model ASICs for scientific and engineering simulation

Noetic is all about building custom state model ASICs that give scientific and engineering workflows accelerated, consistent compute, state by state.

We take a customer's simulation workflow, quantize or distill a state model for it using our own process, and let that process drive the design of the silicon. We design and test in Austin, Texas and assemble in Electronic City, Bangalore. We ship the hardware with an open source software layer.

The problem

Physicists, engineers, and increasingly AI-driven startups run simulation-heavy work that needs to encode rich visual and physical information quickly and consistently. Today that work runs on general purpose GPUs or on unified-memory edge systems like Jetson. Those systems are fast on average and inconsistent step to step. For simulation, the inconsistency is the real cost. A pipeline that jitters is hard to trust, hard to schedule around, and hard to compare across runs.

The result is that a lot of teams still spend money on physical tests in uncontrolled, non-digital environments because the digital version is not fast or reliable enough to replace them. That is the spend we are going after.

Why state models, and why silicon

State space models (the Mamba family) hold history in a fixed-size state and update it with a fixed amount of work per step. Nothing grows with sequence length. That is a modeling property, and it is also a hardware property: fixed memory, predictable per-step compute, a datapath that can be laid out once and run the same way every time.

Transformers on GPUs do not have this. Their memory footprint grows with context and their latency moves with it. For a solver stepping through time or space, that is the wrong shape.

So the bet is simple. State models are the right model class for consistent simulation compute, and a state model is regular enough that a small team can build an ASIC around it. We think those two facts belong together.

What we build

A custom ASIC per workflow, not a general accelerator.

1. A customer places an order with a specific workflow. Data center heat flow simulation. Plasma physics. Photonics. Quantitative biology. Anything high dimensional where a state model can carry the physics.
2. We quantize or distill a state model for that workflow using our proprietary process, which uses routing and multiple activation layers to hold accuracy while cutting precision and compute.
3. The quantized model informs the ASIC design procedure: what precision the datapath needs, how much state lives on chip, where the HW/SW split sits.
4. We deliver the part with an open source software stack. Because it is open, we expect a community to form around it, which supports customers without us having to staff every integration ourselves.

Who it is for

Next 6 to 12 months: physicists, engineers, amateur AI enthusiasts, and businesses with simulation-heavy work. These are people who already know their workflow and already pay for compute, and who would rather have a consistent part than a faster GPU.

After that: financial simulation and market making, where latency and consistency are the product and hedge funds pay for both. Same silicon approach, tighter latency targets.

Business model

Order first, then design. We do not build inventory of a general part and hope it fits.

- Price range of roughly $750 to $2,500 per custom ASIC design, with additional cost for heavier modifications and more intensive workloads.
- Austin for architecture, RTL, verification, and physical design. Electronic City, Bangalore for assembly and the more experimental testing, which keeps labor and iteration costs down.
- Once a customer's workflow is locked and we are building the same class of part repeatedly, cost per unit drops and margins improve. The first part for a new workflow is the expensive one. The tenth is not.

The pricing only works if the base architecture is shared and the per-customer work stays in quantization, distillation, and configuration. That is a design constraint we hold ourselves to, and it is why the technical plan below spends so much effort on locked interfaces and a versioned ISA.

Technical plan

This is the plan we are executing now, mapped to the state model direction. Coral is our existing RTL and microarchitecture baseline.

Architecture

- Stand up the v1 performance model running against the existing functional model.

- Define the first workloads and performance targets, including the HW/SW split for the state model scan, the routing, and the activation layers.
- Lock down component interfaces so RTL can work independently.
- Version the ISA and microarchitecture so RTL can start on v1 while architecture continues on v2 until feature freeze.
- Build in basic stats, microbenchmarks, and visibility from the outset.

RTL

- Begin with the existing Coral RTL and microarchitecture definitions until the first architecture handoff.
- Relegate any potential reimplementation to members graduating our apprentice program.
- Have a number of contributors focus on quick feasibility and timing estimates, feeding those numbers back to architecture.
- Establish the review flow before things start landing in main.

Design verification

- Audit the existing Coral test benches and identify the largest gaps.
- Verify the current RTL for baseline functionality.
- Set up the functional model as the golden reference, plan randomized instruction testing, and stand up a UVM base.

Physical design

- Get the current Coral design into the Chamber and run first synthesis and place-and-route trials.
- Work through block partitioning, floorplanning, and block ownership.
- Establish a baseline timing and PPA report.
- Use OpenRAM for now while we work on the memory compiler and PDK access.

Inference and software

- Turn the target workflows (heat flow, plasma, photonics, quantitative biology) into actual benchmarks and metrics that architecture can reference.
- Benchmark against the Jetson and unified-memory systems customers are running on today, since that is what we have to beat on consistency.
- Use the Berkeley VLA tapeout infrastructure as a reference for how to scaffold this.
- Plan the open source release alongside the first part, not after it.

Partners and leadership

- Maintain relationships and follow up with AMD, Apple, Optiver, Partcl, and Silicon Labs.
- Continue working with Berkeley, Papermaster, Cooley, and Mike.
- Push on IP and PDK access, and TAC access for licenses.
- Adopt agentic EDA practices (Chip Agents, arXiv 2506.14074) so a small team can carry a full design flow.
- Send the interface-isolation question and compute wishlist to partners.

What done looks like in 6 to 12 months

- v1 performance model running, first workloads and targets defined, interfaces locked.
- Coral baseline verified, RTL review flow in place, first feasibility and timing numbers fed back to architecture.
- First synthesis and place-and-route results and a baseline PPA report.
- A benchmark suite for the target workflows, with Jetson and unified-memory comparisons.
- First customer workflow quantized and distilled through our process and mapped onto the design.
- Bangalore assembly and test setup running.

What we are asking for

- Industry support and backing to carry the team through the first tapeout and the Austin setup.
- Introductions that help with PDK and IP access, and with foundry and shuttle capacity.
- Compute for simulation and verification runs.

What could go wrong

- If per-customer work leaks into silicon instead of staying in software and configuration, the $750 to $2,500 range does not hold. Locked interfaces and the versioned ISA are how we guard against this.
- PDK and IP access can slip. OpenRAM and the existing Coral flow let us keep moving in the meantime.
- State models in scientific and engineering pipelines are still early. We are betting on that adoption, and the benchmark work is how we prove it to customers before they buy.

1 Links to our work (Note: Exists as isolated segments, from our earlier projects, that Noetic is actively working to map out to State Models

- https://github.com/LonghornSilicon/lambda –> Design workflow we are mapping to state models before tapeout
- https://github.com/LonghornSilicon/tt-longhorn-tiu –> caching and compute mechanisms we are modifying for State Model ASICS
- https://worldsim.irislabx.com –> Older simulation framework we are modiyfing and using to influence ASIC design decisions
- https://humiris.ai –> Agent and model routing procedures we are using to improve the workflow informed quantized model
- https://github.com/LonghornSilicon/pe-apprentice –> program for validating our talent and making sure they have the skills and keen attention to detail to match our design and assembly standards.