



SEPTEMBER 2026 • ISSUE 1

Longhorn Silicon Monthly

Our first month

Hi everyone, welcome to the first Longhorn Silicon newsletter. We're building Reef, our student-designed chip based on Google's Coral NPU, and this issue covers what our teams worked on in September.

We received about 340 applications for our first cohort. Since then, we've started general meetings and Sunday workdays in EER, co-hosted Etched on campus, and selected TinyStories 28M as Reef's first workload. Architecture and DV have also met with engineers from Tenstorrent and Amazon to work through their plans.

Each team wrote its own update on their progress, what they're learning, and what comes next. We've included the current chip specs and project timeline in **Reef at a glance**. Some decisions are still open, and we'll keep updating those as the teams test their assumptions.

In October so far, we've had an engineer from IBM join a general meeting to discuss logic design. We also scheduled a hardware tech talk with Hudson River Trading for October 14. Thank you to everyone who's been showing up, asking questions, and helping us get the project started.

October schedule

- **Oct 13:** Minimal testbench (Arch + DV)
- **Oct 14, 7:30 PM:** HRT x Longhorn Silicon hardware tech talk.
- **Oct 16:** Module initialization (Arch)
- **Oct 20:** Architecture v0 specification
- **Oct 25:** Functional simulator (Arch + Inference)

Reef at a glance

Reef is our custom version of Google's Coral NPU, tuned for running small LLMs. It runs TinyStories 28M using GPT-Neo weights quantized to int8. Lambda (N16) is planned for the spring.

— The chip

BASELINE	Google Coral NPU, release M3-2026-04-27 (RISC-V RV32IMF + RVV)
NODE	sky130 on the ChipFoundry double-wide OpenFrame while we wait on TSMC 65nm approval
AREA AND I/O	32.2 mm ² user area inside the padframe, 63 GPIOs
WEIGHT MEMORY	Off-chip HyperRAM, with one port measuring 90 MB/s at 50 MHz
ON-CHIP MEMORY	40 KB TCM (8 KB ITCM + 32 KB DTCM)
COMPUTE	Coral RVV core plus a 16×16 weight-stationary int8 array
CLOCK	16 to 20 MHz at the slow corner
DECODE	Projected 1.65 tok/s on one HyperRAM port, 2.78 tok/s with an 8k vocabulary

The compute plan is still a proposal until Arch spec v1 is done.

— Optimizations under study

- **OliVe**: store weights in 4-bit with 8-bit outliers, then expand them to 8-bit once they're loaded
- **Lookup tables** for softmax, GELU, and layer norm
- **Vocabulary pruning or low-rank factorization**, since most of the weights are in the embedding table
- **KV cache**: cap the context length first, then try quantization-aware compression

— Netlist releases

Each netlist gets tagged as a release in the reef repository. Once it's out, DV runs regressions against it and PD takes it through the flow.

- **NL (Nov 25)**: a stable baseline with synthesis and timing assumptions
- **PFNL (January)**: the real design, with Arch spec v2 implemented
- **FNL (March)**: frozen RTL, ready for signoff

— Road to tapeout

SEP 25	Workload freeze Done TinyStories 28M	ARCH + INF
OCT 13	Minimal testbench Pointer chasing, matmul output and ordering checks	ARCH + DV
OCT 16	Module initialization Modules and the dataflow between them defined	ARCH
OCT 20	Arch spec v0 One document covering every module and open design questions	ARCH → RTL, PD
OCT 25	Functional simulator A PyTorch model runs in our simulator	ARCH + INF
NOV 1	Arch spec v1 Written spec for every module, with no new features after this. ISA and uarch frozen	ARCH
NOV 14	Timing feedback loop RTL and PD report timing on microbenchmarks through January	RTL, PD → ARCH
NOV 23	Regression confirmed Testbenches pass with RTL and PD	ARCH + RTL + PD
NOV 25	1st netlist (NL) RTL passes smoke regression and first PPA numbers go to DV and PD	RTL → DV, PD
DEC 6	Arch spec v2 Bus widths, memories and behaviors fixed	ARCH
JAN	Pre-final netlist (PFNL) Arch v2 implemented in RTL, with full place-and-route and signoff trials	RTL → PD, DV
MAR	Final netlist (FNL) RTL is frozen, DV signs off, and PD closes timing, DRC, and LVS	RTL, DV, PD
APR 15	Foundry submission (est.) Final GDS goes out for tapeout in 2027 (estimated date)	ALL

PFNL and FNL are month targets for 2027, and leads will set exact dates by Arch spec v2. All foundry dates are estimates.

Members

Leadership

Alan Schwartz • Richard Huang • Chaithu Talasila

Architecture

Sahil Gandhi LEAD
Justin Garrigus LEAD
 Vikshar Rajesh
 Akul Laddha
 Orhan Unuvar
 Gokulkrishna
 Yuvesh Banala

RTL

Peter DiSanto LEAD
Aaron Chien LEAD
 Alex Lekhakul
 Samuel Qiao
 Aarush Tulsyan
 Krish Parikh
 Arya Chakrabarti
 Pranav Ullas

DV

Arsh Guntakal LEAD
Jianxiong Li LEAD
 Johan Shyju
 Yogith Loganathan
 Rachit Kakkar
 Tanooj Kanike
 Josh Hiller
 Emad Khawaja

PD

Nathan Mara LEAD
 Krish Singh

Inference

Shrey Birmiwal LEAD
 Ayaan Dhuka
 Kevin Li
 Ravi Tej Guntuku
 Sam Lin
 Jaden George

Apprentices

Architecture

Satvik Chawla
 Chaejeong Kim
 Hailin Chen
 Nila Arunkumar
 Henry Liu
 Thomas Hopper

RTL

Brandon Zhang
 Anna Ledbetter
 Calum Cheah
 Saanvi Pradhan
 Amy Xu
 Yashas Bangalore
 Sanjith Raman

DV

Vraj Thakkar
 Uday Gunturu
 Ashwin Geeni
 Susan Moseley
 Shreya Athur
 Katerina Yang

PD

Arya Upadhyay
 Youssef Anbar
 Hamza Faisal
 Arun Rajendra
 Krishiv Ray
 Arushi Sen
 Ashwin Vijaianand
 Tejas Vanaparthi
 Anikait Gupta

Inference

Sarvajit Jonnalagadda
 Abhiram Bandi
 Amogh Thodati
 Joshua Vigel

AROUND THE ORG

September meetings and workdays



TEAM 01

Architecture

LEADS

Sahil Gandhi • Justin Garrigus



Members of the architecture team after a guest lecture from an industry engineer.

Welcome to the beginning of the new semester! We're ramping up activity right from the start, and busy meeting the demand the rest of the organization requires. Let's jump into it!

What is architecture?

Future AI accelerators begin with an idea. For example, we may suggest hardware improvements on top of existing architectures like adding a vector engine, putting more area into matrix-multiply units, or switching out an outer-product matrix-multiplier with a systolic array. We may also suggest entirely new kinds of computers, like those which completely replace matrix-multipliers with adders (for the benefit of binary neural networks) or those which are capable of overlaying an LLM's SoftMax operation with its preceding GEMM (in FlashAttention).

While these ideas may be theoretically sound, RTL is expensive to write. We do *not* want to develop and verify RTL for ideas that end up not being worth the effort. Thus, there are several questions which must be answered before any RTL is written:

1. **How good is the idea?** (And what metrics define what "good" means?)
2. **Is the idea better than the alternatives?** (And what even *are* the alternatives?)
3. **Is the idea practical to implement?** (An excellent idea giving 100x speedup may be impossible to develop in practice!)

Regardless of how nice the idea sounds, it will not become *more* than an idea unless each question has an answer... but how do we get an answer? This is where the architecture team comes into play!

Models

Techniques impacting model accuracy can be developed without knowing what the hardware will look like; we explore these by creating evaluation environments in PyTorch, varying things like quantization degrees, layer counts, and attention width, and scoring the responses the LLMs generate. Currently, this is handled in Longhorn Silicon's Inference team.

However, most of our efforts will be in judging *hardware techniques*. Once we know the LLM architecture, our goal is to determine which hardware configuration runs it with the highest efficiency. We do

this by constructing a *performance model*—one which is capable of varying micro-architectural qualities, like the core clock rate, the number of matrix-multiply units, the memory bandwidth, and many more—and running wide *sweeps* across different LLM inputs and environments.

Doing this instills confidence in our proposed techniques (much more confidence than we would garner with just an *idea* written in prose), and helps the RTL team implement the hardware with the assurance that their efforts will not be wasted on an inefficient design.

Accuracy

Although, a constant concern of ours will be in ensuring our created models are *accurate*. After all, a model which suggests an architectural change is beneficial when it really has significant hidden costs will be just as useful as no model at all. To fix this (especially given our extremely-short deadlines), our efforts are focused on emphasizing *trends* just as much as absolute latency predictions. A trend will generally show how one variable depends on a set of others (for example, whether increasing the size of a systolic array *actually* improves compute throughput, given a set of parameters estimating the memory bandwidth, execution-unit latency, and more) without necessarily requiring each parameter be accurate. If the trends hold true for any reasonable parameter, then we can take the trend as fact without needing to have confidence in the values of the parameters themselves.

This allows us to design performance models that are highly modular, internally consistent on the module level, and still useful even in highly uncertain environments. Our group is capitalizing on this by assigning each member ownership on a particular module in the performance model, leading to rapid, risk-aware development!

Goals

Architecture is one of the most fast-paced teams in the group right now. In order for RTL, DV, and PD to actually *make* the hardware, they need to know *what* to make! This means our first goal is to design the **v0 spec**: our initial proposal to the rest of the club on what we believe the hardware will look like. Next month, we'll share our open-source v0 spec and jump forwards into creating our performance model. See you then!

TEAM 02

RTL

LEADS

Peter DiSanto • Aaron Chien



The RTL Team on Workday 1.

Highlights

September was about understanding the existing Google Coral NPU design and preparing the tools and workflows needed to turn Reef's Architecture specifications into synthesizable RTL. This early work is helping us establish a consistent foundation for implementing and evaluating future design changes.

- **Coral NPU RTL flow:** Tracing module hierarchy, interfaces, build dependencies, and synthesis boundaries to understand the path from source RTL to implementation.
- **HLS vs. Chisel:** Comparing productivity and abstraction with hardware control, generated-RTL transparency, verification effort, and ASIC-flow integration.
- **SRAM case study:** Exploring OpenRAM-generated SRAM macros and behavioral models to establish early memory functionality and physical-cost baselines.
- **Block-level synthesis:** Working toward repeatable synthesis of individual modules to assess resource usage, timing behavior, and potential bottlenecks.

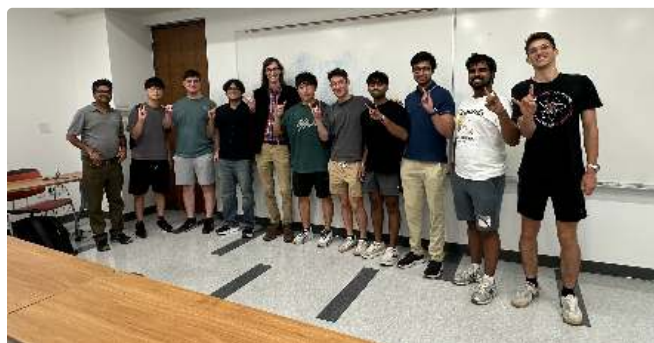
Project update

Our priority is building a dependable RTL-to-synthesis foundation ahead of the Architecture handoff and V1 Specification. By mapping the baseline Coral design, studying memory implementation, and establishing block-level synthesis workflows now, we aim to make incoming architectural changes easier to implement and evaluate. This preparation will also help RTL provide timely area and timing feedback to Architecture and Physical Design as Reef progresses toward its first netlist.

Establishing repeatable synthesis baselines will also make it easier to compare design revisions, identify potential implementation challenges early, and coordinate changes across teams as the specification evolves.

What's next

- **October:** Prepare for module initialization (Oct 16), the ISA and microarchitecture freeze (Oct 20), and Architecture Spec v1 (Nov 1). These milestones will guide the team's initial RTL implementation work.
- **November timing feedback:** Begin the RTL-PD timing feedback loop (Nov 14), using synthesis results to identify timing and area concerns as design blocks take shape.
- **First netlist release:** Work toward the first tagged netlist release (Nov 25), supporting smoke regressions and initial power, performance, and area estimates for DV and PD.



The Longhorn Silicon leads.

TEAM 03

Design Verification

LEADS

Arsh Guntakal • Jianxiong Li



The Design Verification team.

— Highlights

September was primarily focused on preparing for verifying Reef, our custom NPU based on Google's Coral design. While Architecture team worked on defining the initial subsystem specifications and the RTL team established the foundation for hardware implementation, our DV team collaborated closely with both teams and focused on establishing a verification plan to ensure the chip produces expected functional behavior with reliability.

- **Understanding the Coral NPU architecture:** Analyzing the baseline Google Coral architecture, hardware interfaces, and datapath and control path to identify key functionality and potential verification challenges.
- **Verification planning:** Defining how to verify features in inference arithmetic and computation, memory operations, and controllers.
- **Verification infrastructure:** Developing a reusable simulation and regression environment that can support IP-level and subsystem-level testing as Reef evolves and scales up.

— Project update

Our priority this month is to understand the Coral and Reel architectures while consolidating our fundamentals in UVM/SV basics, in-order and out-of-order processor architectures, programming and algorithms, and Verilog RTL.

Rather than waiting until the design is complete, we aim to develop verification alongside with Architecture and RTL teams so that features and corner cases can be identified early.

As the architecture becomes more stable, we plan to verify from bottom to top. We will begin with individual IP blocks. After when functional coverages reach milestones, we will scale the verification up so that it will be helpful for debugging. We will expand from module tests to end-to-end verification, making sure computation, memory, and controller work together correctly.

— What's next

- **October:** Architecture and specifications: Study the developing architecture and module specs, identify important verification requirements, and prepare for initial testbench development.
- **November:** Verification plan and UVM architecture: Draft the verification plan, define testbench architecture, and establish functional coverage goals for individual IP blocks.
- **Toward tapeout:** Develop UVM components, design directed and constraint random tests, debug RTL issues, and run regression tests on tagged releases. Gradually improve functional coverage, achieve coverage closure, and work toward functional signoff before tapeout.



The Longhorn Silicon group.

TEAM 04

Physical Design

LEAD
Nathan Mara

Our PD apprentices.

— Who we are

Each of our physical design (PD) apprentices owns one block of the Longhorn Silicon Neural Processing Unit (NPU), and they're taking it all the way to tapeout, when the finished design gets sent to the factory. To get ready, they're doing a couple of weeks of training based on a graduate ASIC design course at UT Austin. Our PD flow is being built in the Cadence design suite, the same tools professional chip designers use.

— What is physical design

The RTL team writes a chip's logic as code. The factory needs something else: the exact shapes of every transistor and wire. Physical design gets you from one to the other. We decide how big the block is, place the cells, wire them up, and then check that the result works. A whole chip is too big to lay out in one go, so it's split into blocks that are built separately and joined at the end, and each apprentice gets one. Along the way you pick up algorithms, circuits, computer architecture, and even some materials science and thermodynamics, since heat is a real problem on a dense chip.

— Getting to tapeout

There are ten steps take a design from RTL to tapeout:

BUILD	CHECK
1 Synthesis code to gates	6 Extraction measure wires
2 Floorplan size and power	7 Timing check setup & hold
3 Placement place std cells	8 Power check power use
4 Clock tree clock every cell	9 DRC manufacturability
5 Routing draw the wires	10 LVS check connections

A design is done only when every check passes on the same layout. A timing fix can move cells and break a power check somewhere else, so you rerun everything.

— Trade-offs in the design

Every choice in PD trades off power, performance and area (PPA). Faster chips burn more power and take up more room, and saving power usually costs area. You can't win all three, so you pick where to optimize.

Trade-off	What happens	Cost
Speed vs power	Power goes up with clock speed, and the fastest cells leak the most when idle.	Fast cells and upsized cells and more leaky and consume more power.
Speed vs area	Hitting a speed target means adding buffers and bigger gates.	Larger cells and extra buffers take more space.
Power vs area	Shutting parts of the chip off, or running them at a lower voltage, takes extra cells.	Clock gating and power gating consume extra area.

PPA isn't the whole list either. Wires have to carry current for years without wearing out, and transistors slow down as they age, so we build in margin. A bigger chip fits fewer times on a wafer, which drives up cost and reduces yield. And one pass through routing and the final checks can take hours to days, so every redo eats into the schedule.

— What's next

First we'll finish the intro training. Then we start building a real PD flow, using Google's Coral NPU as a test case. The goal is a fleshed-out flow we can reuse as a template once the RTL team hands us our own design. Along the way we'll compare Coral's early estimates against our targets and feed what we learn back to the architecture and RTL teams, so problems show up before the design is locked in.

TEAM 05

Inference

LEAD
Shrey Birmiwai

The inference team.

Howdy! We're excited to share what we do and what we're working on.

What is inference?

When you talk to Gemini or ChatGPT or Claude, it's running a program on hardware to convert user prompts into tokens and finally into output text. When Large Language Models (LLMs) first started becoming popular, inference was a second thought, as improving models was far more important than making them run efficiently. Now, as models are getting larger (10T params or more!), hardware getting increasingly more expensive (companies spending \$100B for datacenters!), and margins shrinking, running models on hardware efficiently is more important than ever. In the real world, inference involves designing systems that with complicated hardware (for example, multi node distributed GPUs) and writing efficient kernels that can utilize your hardware most optimally. Every incremental improvement in hardware utilization can save billions of dollars, and that's exactly why frontier labs pay huge sums of money to kernel engineers!

What's the inference team at Longhorn Silicon about?

Currently, even while using the best hardware and the best software, it is very hard to reach the theoretical limit of the hardware. This is because the software and the hardware were not fully codesigned together.

Longhorn Silicon believes in radical codesign. The inference team works closely with the hardware teams to provide feedback on what hardware systems can be designed differently for us to have more efficient hardware for LLM serving. Additionally, we work with the hardware teams to utilize the hardware we have to push performance.

What are we working on?

Model efficiency

The goal of this group is to take existing large language models and make them more efficient for our hardware. By trying tricks such as KV re-ranking, low-rank factorization, speculative decoding, vocab size reduction, and more, we hope to push the frontier of model quality per unit of efficiency. In our initial experimentations on simulators, we foresee the Reef chip to have memory bandwidth as the primary bottleneck, which has led us to explore running 24M parameter GPT-Neo flavored models fine tuned on the TinyStories dataset. Additionally, we are looking into mixed precision quantization, RoPE instead of table lookup, and vocabulary size reduction to reduce the memory footprint of this model to push the "intelligence per byte" frontier.

Kernels

The kernel group is aiming to utilize the hardware to the maximum amount possible. This involves writing the best kernels, building systems to perform tensor parallelism (running models across multiple chips!), pipeline parallelism, etc. Additionally, as we add more custom hardware instructions, this team will be building a compiler that can compile programs to efficiently utilize the hardware.

Profiling

This group works closely with architecture to codesign the hardware with the LLM workload in mind. This group will run various model workloads and profile the hardware utilization and try to find gaps or hardware improvements that can be built for our chips.

Thanks for reading and see you next month! Hook 'Em!

**GET IN TOUCH**

Thanks for reading

We're looking for engineers who can review our work, speak at a meeting, or mentor our members, and partners who can help support the tapeout. If you're interested in working with us or want to hear more about Reef, email us below.

info@longhornsilicon.com

longhornsilicon.com

github.com/LonghornSilicon