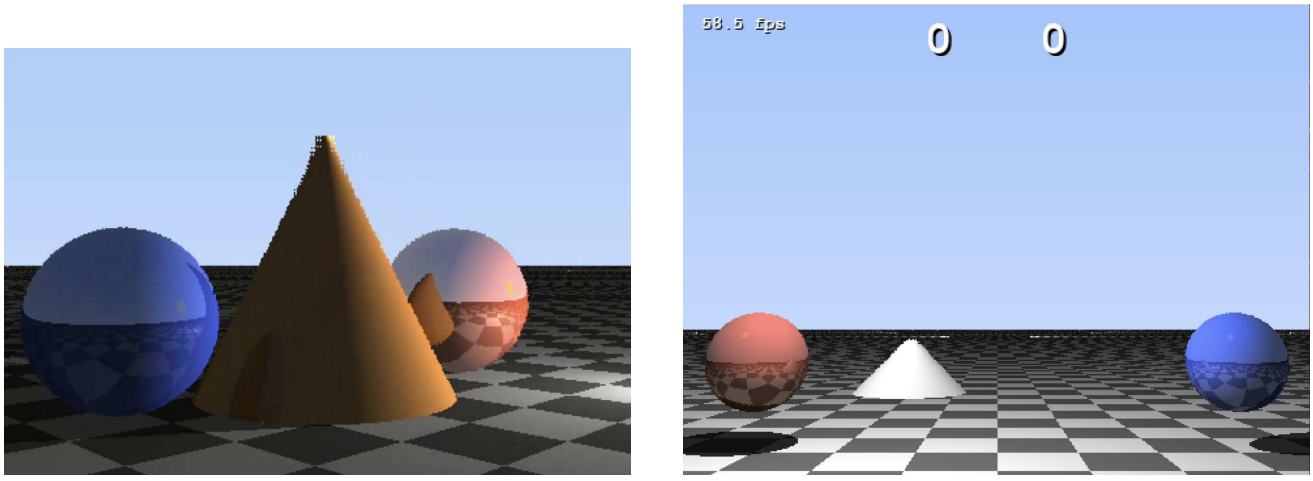
Hardware Raytracer on the DE1-SoC
CSEE 4840 Final Project Report

Matthew Lou Tony Giannini Innokentiy Kaurov

May 2026

Abstract

This project implements an interactive fixed-point raytracer on the DE1-SoC. A desktop client sends camera and scene parameters over TCP to software running on the ARM HPS. The HPS writes those parameters into a custom Avalon-MM peripheral, starts a full-frame render, and streams the finished 480 by 360 BGRX frame back to the client. The FPGA traces primary and reflected rays through two spheres, a finite cone, a floor plane, a sky gradient, a sun billboard, shadows, and configurable floor/surface colors. The main hardware contribution is a four-lane, fixed-initiation-interval ray pipeline with a row-level DMA engine that writes completed pixels directly into HPS DDR3. The main software contribution is a Linux driver and server/client protocol that expose the accelerator as a double-buffered frame producer with only one frame-start write and one wait ioctl per frame. The completed system renders the scene at over 60 frames per second on a 125 MHz `divclk`, an end-to-end improvement of more than 60× over the unpipelined SystemVerilog baseline, a direct port of the reference algorithm running on the same FPGA without parallelism, DMA, or reflections, which produces $1-2$ frames per second. The deployed design uses 85% of the Cyclone V's ALMs and 69% of its DSP blocks.



(a) Default scene. (b) Pong demo (FPS overlay top-left).

Figure 1: Representative 480×360 frames rendered live by the hardware and streamed to the desktop client over TCP. (a) Two reflective spheres, a finite Y-axis cone, a checker plane, sky gradient, and a sun-tinted horizon — exactly the bytes that arrive at the client. (b) The Pong demo: red and blue spheres act as joystick-driven paddles, the cone is the ball, scene physics runs on the desktop client while rendering stays on the FPGA. The top-left overlay reads 57.8 fps, consistent with the steady-state numbers in Table 5.

Contents

1 Introduction

The goal of the project was to make raytracing interactive on the DE1-SoC while preserving the structure of a real hardware/software embedded system. The design deliberately splits the workload. Software performs infrequent, frame-level tasks such as joystick input, camera yaw/pitch trigonometry, scene packing, network I/O, and cache maintenance. Hardware performs the repeated per-pixel work: ray generation, primitive intersection, shading, shadow tests, reflection spawning, color accumulation, and DMA output.

The final system renders a 480 by 360 scene. The configurable scene contains two spheres, a finite vertical cone, a floor plane with multiple procedural patterns, a sky gradient, a sun disk derived from the light vector, Lambertian diffuse lighting, ambient shadow fallback, hard shadows, and recursive mirror-like reflections. The accelerator is exposed to Linux as `/dev/raytracer`. Userspace maps a 4 KiB CSR (Control and Status Register) page and a two-slot frame ring in DDR3, sends a packed frame-input structure to the hardware, waits for completion through an ioctl, and returns the completed frame to the desktop viewer.

1.1 Project Overview — Revised From the Proposal

The original proposal described a much smaller system. The scene was supposed to contain one of three discrete primitives — a sphere, a cube, or a pyramid — chosen by board buttons; the only animated quantity was the light position, controlled by a joystick on the HPS; the host was supposed to talk to the FPGA over a custom GPIO batch protocol; and the success criterion was simply that the FPGA show a clear acceleration benefit over a host-only renderer, possibly with progressive updates rather than a full real-time frame rate. The proposal also pencilled in a 24-bit Q14.10 fixed-point word and six independent four-stage pipelines (ray gen, intersect, shade, shadow/reflect).

The delivered system differs from that plan in several concrete ways. Each change was driven by a specific implementation lesson:

- **Scene primitives.** We replaced the cube and pyramid with a second sphere and a finite vertical cone. A sphere and a cone share a common quadratic intersection structure ($at^2 + 2bt + c = 0$), so a single fixed-point reciprocal/square-root machine handles both. A cube would have required six plane equations with slab tests, and a pyramid four; both expose more discontinuous normals than a smooth surface and reflect less interestingly. The cone also turned out to be the most visually distinctive primitive in the demo.
- **Multiple primitives at once.** The original button-selected single-object scene became a fixed multi-object scene of two spheres, a cone, and a floor. The motivation was that reflections only become visually meaningful when one primitive can be seen reflected in another. The configurable scene state lives in roughly 50 Avalon-MM CSRs.
- **Host interface.** GPIO was replaced with TCP over Ethernet. The driving factor was bandwidth: a 480×360-pixel frame is roughly 700 KB at four bytes per pixel, which cannot be shipped over GPIO at interactive frame rates but is comfortable on the DE1-SoC's Gigabit Ethernet port. As a side benefit, the desktop client and the HPS server were able to evolve completely independently of the FPGA build.
- **User input.** The joystick now controls the camera rather than the light. Moving the camera through a static-lit scene was substantially more useful for debugging shading and reflections than

moving the light through a static viewpoint, and it also exposed the camera-basis CSRs, which the proposal did not anticipate. The light still moves in scripted demos.

- **Fixed-point word.** The earliest builds used Q14.10 in a 24-bit word. After running into precision artifacts on the floor plane at grazing angles, the word was widened to 27-bit Q14.13. The pipeline and the SW header track the same parameter so the change was one synchronized edit across hardware and software.
- **Lane count.** The proposal envisioned six independent pipelines. In practice $K = 4$ gave the best timing/area trade-off; the K-way arbiter and the single-port color accumulator have rising fan-out costs above $K = 4$. We kept the original idea of fixed-II staggered lanes; only the number changed.
- **Performance.** The proposal allowed progressive updates and did not require a target frame rate. The final system reaches 60+ fps at 480×360 with reflections enabled, fast enough to feel responsive to joystick input.

These choices follow the same principle each time: spend hardware on the inner loop (per-pixel ray math) and spend software on the outer loop (camera matrix, scene packing, networking). What we removed — discrete primitive selection, GPIO — belonged to neither loop cleanly and would have spread complexity across both.

2 System Overview

2.1 Top-Level Architecture

The system has six major pieces: a desktop client, a TCP server on the HPS, a Linux kernel driver, the Avalon interconnect generated by Platform Designer, the custom raytracer peripheral, and HPS DDR3 used as the frame destination.

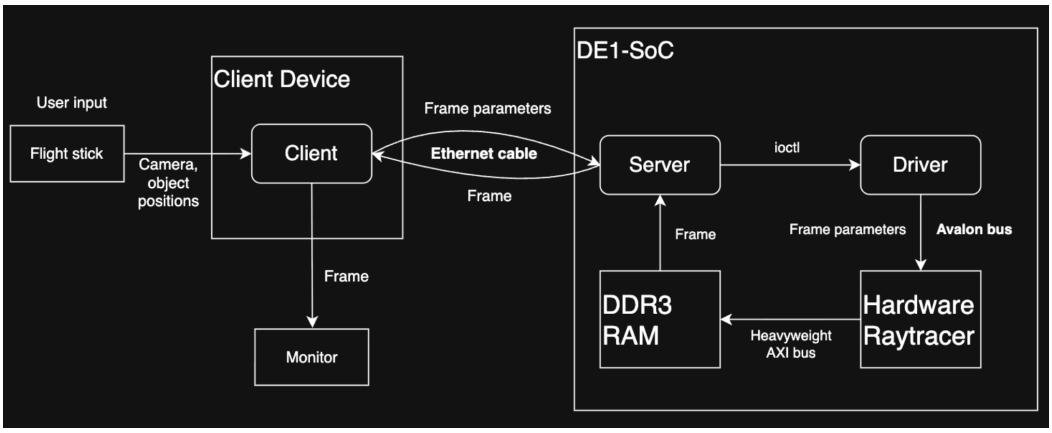


Figure 2: End-to-end system. The desktop client reads joystick input and sends 196-byte frame-parameter packets over Ethernet to the HPS server. The server stages the parameters into the raytracer peripheral through the kernel driver and the Avalon bus, kicks a frame, and waits for the FPGA to complete it. The FPGA writes the finished 691,200-byte BGRX frame into the HPS DDR3 frame ring via a heavyweight AXI burst master; the server reads the frame back through the cached mmap and ships it to the client over the same TCP connection. The client decodes it to the monitor.

The key throughput decision is that software does not read pixels from CSR registers. A full 480 by 360 frame at four bytes per pixel is 691,200 bytes. Reading that through a register aperture would make the HPS a per-pixel bottleneck. Instead, hardware fills a row-sized on-chip color buffer, then a DMA master writes that row into a contiguous HPS DDR3 frame slot. The driver maps two full frame slots to userspace so the HPS can read one frame while the FPGA writes the other.

2.2 Frame-Level Data Flow

At connection time the desktop client sends two frame-input packets to prime the server-side pipeline. For each steady-state iteration, the HPS waits for frame N, immediately starts frame $N + 1$, sends frame N to the desktop, and receives inputs for frame $N + 2$. The pipeline depth of two hides most of the network and userspace copy time behind the next hardware render.

1. The desktop client computes the camera basis from yaw and pitch and packs 49 network-order 32-bit words, or 196 bytes, of scene parameters.
2. The HPS server receives the packet and writes the corresponding CSRs through the mmaped register page.
3. A write to `CONTROL[2]` starts a full-frame render. Software does not kick each row.
4. The hardware frame FSM walks all 360 rows. Each row is one batch of $N = 480$ pixels.
5. The batch pipeline writes a completed row to one of two M10K-backed color banks. The DMA master bursts that bank into HPS DDR3 while the next row is being computed in the other bank.
6. The frame FSM waits for the last DMA burst, toggles the frame `write_slot`, and asserts `FRAME_DONE`.
7. The driver's `RTB_IOCTL_WAIT_FRAME` clears the sticky done bit, invalidates the finished frame slot from the CPU cache, and returns the slot index to userspace.

2.3 Clock Domains and Qsys Integration

The Qsys system is deliberately minimal: four instances — a 50 MHz clock bridge (`clk_0`) wired to the board oscillator `CLOCK_50`, the hard-processor system (`hps_0`), an integer-N PLL (`pll_0`) that multiplies the 50 MHz reference up to 125 MHz `outclk0`, and the raytracer IP (`raytracer_0`). The raytracer is the only instance in the system clocked off the 125 MHz output; the HPS-to-FPGA Avalon bridges (`h2f`, `h2f_lw`, `f2h_axi`) all run on the HPS-supplied 100 MHz `clk_0` clock. Every Avalon transaction between the HPS and the raytracer therefore crosses a clock domain. We use the Qsys-generated handshake clock-crossing adapter rather than custom synchronizers; the adapter is a few extra cycles of latency per transaction, but it is a per-frame cost (one CSR-write burst on kick, one polled status read on wait) so it does not affect steady-state throughput.

The 50 MHz reference and the 125 MHz `divclk` are both created as named clocks in the SDC file. STA is told about the clock-crossing through the auto-generated HANDSHAKE constraints, so we did not have to write manual false-path or multi-cycle path entries. The only timing path the design needs to close is the 8 ns period inside the `divclk` domain, which is what Section **??** discusses in detail. One practical note: editing the Qsys system and re-running `qsys-generate` has on at least one occasion silently produced a non-functional PLL output, which compiles and "fits" but renders the design unclocked. The safer iteration loop is to confine edits to `raytracer.sv` and avoid Qsys-level changes unless absolutely necessary.

3 Rendering Algorithms

3.1 Fixed-Point Representation

The hardware uses signed 27-bit fixed point with 13 fractional bits, i.e. Q14.13. The value 1.0 is represented as 2^{13}. This width was chosen to fit coordinate ranges such as camera $z = -8$, scene bounds, intermediate products, and Cyclone V DSP behavior. Multiplication produces a double-width product and returns bits $[WORD - 1 + FRAC\_BITS : FRAC\_BITS]$, so the result stays in Q14.13. The hardware avoids general floating point entirely.

3.2 Camera Rays

The desktop computes the camera basis vectors $\mathbf{r}$, $\mathbf{u}$, and $\mathbf{f}$ from yaw and pitch. Sending the basis instead of angles avoids trigonometric units in hardware. For a pixel (x, y), the screen coordinates are

$$p_x = \left(\frac{2x}{W} - 1\right)\frac{1}{2}, \qquad p_y = \left(1 - \frac{2y}{H}\right)\frac{3}{8},$$

where $W = 480$ and $H = 360$. The primary ray direction is

$$\mathbf{d} = \mathbf{f} + p_x\mathbf{r} + p_y\mathbf{u}.$$

This direction is not normalized before intersection. The sphere and cone intersection equations carry $\mathbf{d} \cdot \mathbf{d}$ explicitly, which saves a normalization divider.

3.3 Sphere Intersection

For a sphere centered at $\mathbf{C}$ with radius R, points on the ray are $\mathbf{o} + t\mathbf{d}$. Let $\mathbf{oc} = \mathbf{o} - \mathbf{C}$. Substituting into $\|\mathbf{p} - \mathbf{C}\|^2 = R^2$ gives

$$at^2 + 2ht + c = 0, \quad a = \mathbf{d} \cdot \mathbf{d}, \quad h = \mathbf{d} \cdot \mathbf{oc}, \quad c = \mathbf{oc} \cdot \mathbf{oc} - R^2.$$

The hardware uses the half-b form. The discriminant is $\Delta = h^2 - ac$, and the near root is

$$t = \frac{-h - \sqrt{\Delta}}{a}.$$

If $\Delta < 0$, or if the ray is a reflection spawned from the same primitive, the sphere is marked as no-hit.

3.4 Plane and Cone Intersection

The floor is the plane $y = 0$. Its intersection is

$$t_{plane} = \frac{-o_y}{d_y}.$$

The denominator is clamped near zero so horizon rays do not overflow the fixed-point range.

The cone is a finite, vertical, downward-opening cone with apex $\mathbf{A}$, height H_c, and $k^2 = \tan^2\theta$:

$$(x - A_x)^2 + (z - A_z)^2 = k^2(y - A_y)^2, \quad A_y - H_c \le y \le A_y.$$

Let $\boldsymbol{\delta} = \mathbf{o} - \mathbf{A}$. Substitution produces

$$a_c t^2 + 2b_c t + c_c = 0,$$

where

$$a_c = \|\mathbf{d}\|^2 - (1+k^2)d_y^2, \quad b_c = \delta_x d_x + \delta_z d_z - k^2 \delta_y d_y, \quad c_c = \delta_x^2 + \delta_z^2 - k^2 \delta_y^2.$$

The hit candidate uses $(-b_c - \sqrt{b_c^2 - a_c c_c})/a_c$ and then passes a slab check on y. The implementation also rejects a small region near the apex to avoid numerical noise and supports a bottom cap through the finite-height logic.

3.5 Normals, Lighting, and Shadows

The surface normal for a sphere is $(\mathbf{p} - \mathbf{C})/\|\mathbf{p} - \mathbf{C}\|$. The plane normal is $(0, 1, 0)$. For the cone, the normal comes from the gradient of the implicit cone equation:

$$\nabla f = (x - A_x, -k^2(y - A_y), z - A_z).$$

On the cone surface its magnitude can be written as

$$\|\nabla f\| = \sqrt{k^2(1+k^2)}\,|A_y - y|.$$

The client precomputes $\sqrt{k^2(1+k^2)}$ once per frame and sends it as `CONE_NORM_FACTOR`, saving a hardware square root in the cone normal path.

For visible surfaces, stage H computes Lambertian diffuse lighting with an ambient term:

$$I = \begin{cases} c, & \text{sky}, \\ c \cdot \frac{5}{32}, & \text{shadowed}, \\ c \cdot \frac{5}{32} + c \cdot \max(0, \mathbf{n} \cdot \mathbf{l}), & \text{lit}. \end{cases}$$

The shadow stage casts a secondary ray from a biased hit point toward the light and tests both spheres and the cone. A primitive blocks the light if the shadow ray hits it at positive t before the light distance. The design uses an algebraic, sign-aware cone shadow comparison so that the cone path does not require an additional reciprocal unit.

The sky color is a vertical gradient. The sun is modeled as a billboard at infinity aligned with the light vector. Instead of normalizing $\mathbf{d}$ and $\mathbf{L}$, the hardware tests

$$(\mathbf{d} \cdot \mathbf{L}) > 0 \quad \text{and} \quad (\mathbf{d} \cdot \mathbf{L})^2 \geq \frac{63}{64}\|\mathbf{L}\|^2\|\mathbf{d}\|^2.$$

The top-level wrapper computes $K_{\text{sun}} = (63/64)\|\mathbf{L}\|^2$ continuously from the light CSRs using one shared multiplier and a shift-subtract.

3.6 Recursive Reflections

When a ray hits a reflective object, the reflected direction is

$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}.$$

Each ray carries a side-channel metadata struct with three fields: recursion depth, a self-intersection exclusion mask, and cumulative attenuation shift. A child ray is spawned only if the hit primitive is reflective, the depth is below `MAX_DEPTH`, and the accumulated shift would stay below 8. The color accumulator stores per-primary-ray partial sums. Each emitted contribution is shifted by the cumulative attenuation; the final color is written only when the reflection chain terminates.

4 Hardware Design

4.1 Custom Peripheral

The Platform Designer component `raytracer_batch_mm` has an Avalon-MM slave for CSRs and debug reads, plus an Avalon-MM master for DMA writes into HPS DDR3. The wrapper contains CSR registers, a frame FSM, a two-bank row color buffer, the DMA master, and the compute core `rtl_batch_pipe`.

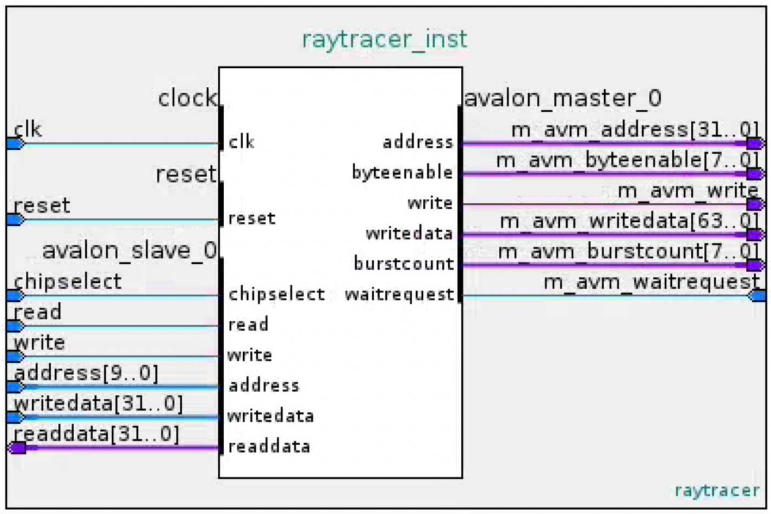


Figure 3: The `raytracer_inst` block as exposed to Platform Designer. `avalon_slave_0` (left side: `chipselect`, `read/write`, `address`, `writedata`, `readdata`) carries CSR traffic from the HPS into the peripheral. `avalon_master_0` (right side: `m_avm_address`, `m_avm_byteenable`, `m_avm_write`, `m_avm_writedata`, `m_avm_burstcount`, `m_avm_waitrequest`) is the DMA path out to the HPS DDR3 frame ring via `f2h_axi_slave`. Both ports run on the same 125 MHz `clk`; Qsys handles the clock-domain crossing to the 100 MHz HPS bridge clock.

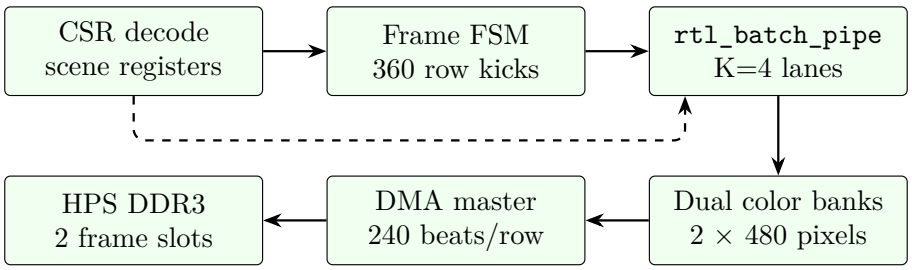


Figure 4: Raytracer peripheral internals. Solid arrows show the row-processing path; the dashed arrow shows scene state broadcast from the CSRs into the compute core. Compute and DMA overlap through ping-pong color banks.

4.2 Module Hierarchy

The entire raytracer fits in a single SystemVerilog file (`raytracer.sv`, ≈ 4000 lines) plus a small package (`raytracer_pkg`). The synthesis top is `raytracer_batch_mm`; the rest of the modules are leaf primitives or pipeline stages. Module instantiation is summarized below.

Table 1: Module instantiation hierarchy. Quartus prunes unused fields of the `scene_t` struct at each consumer, so passing the whole struct through every stage costs no additional wires compared to passing per-primitive structs explicitly.

Module	Instantiates
`raytracer_batch_mm` (top)	one `rtl_batch_pipe`, one `rtl_dma_master`, one `sun_threshold_compute`; CSR decode, frame FSM, two-bank color memory.
`rtl_batch_pipe`	$K = 4$ instances of `rtl_pipe` with staggered `PHASE` offsets, one shared `basis_compute`, the K-way issue arbiter, and the color/accumulator memories.
`rtl_pipe`	one of each: `rtl_ray_gen`, `rtl_isect_prep`, `rtl_hit_resolve`, `rtl_normal_div`, `rtl_reflect_dir`, `rtl_light_dir_div`, `rtl_shadow_test`, `rtl_shade_blend`; per-pipe `scene_local` relay.
Per-stage modules ($\times 8$)	tick-muxed `fp_mul` instances (registered to DSP output FFs) plus `fp_sqrt` or `fp_inv` where the stage needs one.
`basis_compute`	five DSP-mapped multiplies for the per-frame camera basis; combinational adder tree.
`rtl_dma_master`	Avalon-MM master FSM (5 states): `DMA_IDLE`, `DMA_PREFETCH`, `DMA_WRITE`, `DMA_COMMIT`, `DMA_DONE`; latches `addr_burst` and `dma_bank_sel` at `DMA_IDLE`→`DMA_PREFETCH`.
`sun_threshold_compute`	one DSP time-muxed across three multiplications to produce $K = (63/64)\|\|\mathbf{L}\|\|^2$ once per several cycles.
`fp_mul`, `fp_sqrt`, `fp_div`, `fp_inv`	primitive arithmetic units; `fp_mul` is single-cycle with a DSP output FF, the others are bit-serial iterative.

4.3 Pipeline Core

The compute core renders one row per batch. It instantiates four `rtl_pipe` lanes with staggered phases $PHASE = II \cdot i/K$. The initiation interval is 43 cycles, chosen to contain the 41-cycle fixed-point reciprocal/divider plus setup and capture margin. Staggering ensures that at most one pipe issues a new ray or writes the shared accumulator/color memories in a given cycle. Each per-pipe stage module is parameterized by `FRAC_BITS=13`, `WORD=27`, and `TAG_W=7`; signal widths are `WORD=27` bits for scalars, 1 bit for flags, 3 bits for `hit_type`, and three scalar fields for each three-component vector (three discrete wires rather than a packed vector, so Quartus can place each component near its consumer). Each ray also carries an 11-bit `ray_meta_t` side band (3-bit `depth`, 3-bit one-hot `excluded`, 5-bit `total_shift`).

Table 2: Per-pipe stage role and payload emitted into the next stage. Stage offsets inside the 43-cycle II are fixed by construction; the spawn-slot writeback happens at the H emit tick (`tick=15`). `pipe_baggage_t` (`sv`, `tag`, `ray_meta_t`) rides alongside every payload.

Stage	Module	Role and output payload
A	`rtl_ray_gen`	Selects primary or reflected direction, computes $\|\|\mathbf{d}\|\|^2$, and precomputes the cone-a coefficient. Emits `raygen_raw_{x,y,z}`, ray origin, and the pipe baggage.

Stage	Module	Role and output payload
B	`rtl_isect_prep`	Forms sphere and cone quadratic terms, starts square roots and reciprocals, prepares the plane denominator. Emits per-sphere $\{n_b, \Delta\}$ + `nh`, cone $\{n_b, \Delta\}$ + `nh`, $1/d_y$ for the plane, and $\mathbf{o}, \mathbf{d}, a, 1/a$ pass-through.
C	`rtl_hit_resolve`	Selects the nearest sphere/plane/cone/sky hit, reconstructs the hit point, picks the base color, and starts normal-magnitude work. Emits nearest $\mathbf{p}$, normal precursor `df`, light precursor `tl`, base $\{R, G, B\}$, `hit_type`, $\|\texttt{df}\|^2$, $\mathbf{d}$ pass-through.
D	`rtl_normal_div`	Computes $1/\|\texttt{df}\|$ for normal reconstruction; rides forward everything else.
E	`rtl_reflect_dir`	Recovers the surface normal $\mathbf{n}$, computes the reflection vector $\mathbf{r}$, kicks the light-distance computation.
F	`rtl_light_dir_div`	Computes $1/\|\mathbf{L} - \mathbf{p}\|$ so the next stage can materialize the unit light direction.
G	`rtl_shadow_test`	Casts the shadow ray and runs the sphere/sphere/cone occluder test; forwards the reflected-ray side band.
H	`rtl_shade_blend`	Computes ambient/diffuse shading, handles sky/sun and shadow cases, clamp01 the result, and emits the final $\{R, G, B\}$ at `tick=15` along with the (hit, refl) tuple that the spawn router needs.

The eight stages are instantiated inside `rtl_pipe` as `u_raygen`, `u_isect`, `u_hit`, `u_norm`, `u_refl`, `u_ldir`, `u_shdw`, `u_shade`; `rtl_batch_pipe` replicates `rtl_pipe` $K = 4$ times with staggered $PHASE$ offsets and shares one `basis_compute` across the four lanes. A primary ray takes approximately $8 \times 43 + 15 = 359$ cycles from issue to color emission. In the ideal no-reflection case, four lanes issue 4/43 pixels per cycle. At 120 MHz this is about 11.2 Mpixels/s, so 172,800 primary pixels have a compute lower bound of about 15.5 ms per frame before reflection rays, row draining, CPU cache invalidation, and TCP transfer. At the configured 125 MHz clock the lower bound is about 14.9 ms. Timing closes at 125 MHz with +0.200 ns of positive slack on the `divclk` domain, so the design runs at its constrained period and does not require derating.

4.3.1 Initiation Interval and Phase Staggering

The initiation interval $II = 43$ is set by the bit-serial `fp_div` module, whose latency at $WORD = 27$ and $FRAC\_BITS = 13$ is $WORD + FRAC\_BITS + 1 = 41$ cycles; $II = 43$ adds two cycles of capture margin so that a new operand can never collide with a previous one in flight. The four pipes are PHASE-staggered at $PHASE_i = \lfloor II \cdot i/K \rfloor = 0, 10, 21, 32$ inside the 43-cycle window. The staggering has two purposes:

- The K-way issue arbiter is a one-hot scan: in any cycle at most one pipe is at its `at_load_tick` signal (where `tick == II-1`), so two pipes never request issue on the same edge. This eliminates the need for an issue FIFO or true arbitration — the priority encoder is just a structural guarantee.
- The shared reflection-accumulator memory and the color BRAM are similarly write-conflict-free. The drop tick is also one-hot across pipes (offset by a constant from the load tick), so each memory's single write port sees at most one writer per cycle.

4.3.2 Issue Arbiter and Spawn Routing

`rtl_batch_pipe` contains a small combinational arbiter (`raytracer.sv:3668-3700`) that decides each cycle which (if any) pipe is given a ray and where that ray comes from. The arbiter scans pipes in priority order: for each pipe $p \in \{0, \ldots, K-1\}$, if pipe p is at its load tick and the batch is still busy, then pipe p is the unique issuer for this cycle. The scan stops on first match (`!issue_fire` guard), so two pipes can never issue on the same edge.

Each pipe owns a single-entry `spawn_slot` (raytracer.sv:3659–3660). The slot is a packed `spawn_entry_t` struct holding the child ray's tag, origin (= parent hit point), direction (= reflection vector), and updated `ray_meta_t` (depth +1, the parent's `hit_type` as a 3-bit one-hot `excluded` mask, and `total_shift` incremented by the parent primitive's `reflect_shift`). The spawn slot is written by the same pipe's stage H whenever it emits a reflection-eligible hit; the arbiter consumes it on that pipe's next load tick.

Three details matter:

- **Spawn preference over primaries.** When pipe p is at its load tick, the arbiter checks `spawn_slot_valid[p]` first. A pending reflection child is always issued before the next primary ray. This keeps reflection chains from queueing up behind unrelated primary work and bounds the per-pipe in-flight depth at one reflection.
- **Primary ray counting.** `issue_count` tracks how many primary rays the current batch has issued. It increments only when the arbiter issues a primary (`issue_fire && !issue_is_spawn`, raytracer.sv:3948). Spawn issues re-use the parent's tag and never advance the counter. The arbiter blocks new primary issues once `issue_count == BATCH`; remaining work is reflection drain.
- **Batch completion.** `complete_count` increments on every chain-terminal emit (i.e. on a ray that does not spawn a child). When `complete_count == BATCH` the row is finished. Because reflections do not increment either counter (they consume one slot and produce one slot), the bookkeeping is one ray in, one ray out per primary regardless of reflection depth.

An earlier design used a multi-entry FIFO for spawn buffering; it was replaced with the single-entry per-pipe slot (commit `1455294`) because the staggered $II = 43$ schedule already guarantees one spawn-or-primary issue per pipe per II cycles, so deeper buffering bought nothing and cost a measurable amount of fan-out on the arbiter's priority encoder.

4.3.3 Reflection Accumulator

Reflections are recursive in algorithm but flat in hardware. `rtl_batch_pipe` owns a single shared accumulator memory `acc_mem[BATCH]` of type `lane_acc_t = {pending, r[15:0], g[15:0], b[15:0]}` (raytracer.sv:3743–3788), indexed by the pixel-within-batch tag and shared across all K pipes. The accumulator is the rendezvous point for partial sums that arrive from any pipe over multiple reflection bounces; the phase staggering guarantees that at most one pipe drops a ray (and therefore reads/writes the accumulator) on any given cycle.

1. When a primary ray is issued, its accumulator entry is implicitly zeroed by the `pending=0` state.
2. Each time a ray terminates (i.e. the pipe drops it at `tick=15` and does *not* spawn a child), its contribution is added into the accumulator. The contribution is the stage-H color shifted right by `ray_meta_t.total_shift`, which is the cumulative bit-shift attenuation along the reflection

chain so far. A drop that does not spawn writes the new sum back and asserts `acc_emit_w`, which routes the pixel to the color BRAM.

3. Each time a ray spawns a child, the child's `ray_meta_t.total_shift` is set to the parent's `total_shift` plus the hit primitive's `reflect_shift` (a 5-bit CSR-programmed power-of-two attenuation per primitive: 0 disables reflection, 1 keeps 50%, 2 keeps 25%, etc.).

4. The spawn decision in stage H is gated by three conditions (`rtl_pipe.sv:3370-3376`): the parent's depth is less than `max_depth_r` (the CSR `MAX_DEPTH`), the hit primitive is reflective (`reflect_shift != 0`), and the prospective new `total_shift` would stay below 8 bits. The third gate exists because the per-channel accumulator is 16 bits wide above the 8-bit base color; a child contribution shifted by ≥ 8 would be lost anyway, so the spawn is suppressed and the parent terminates at its current depth.

This makes reflection a strictly local hardware operation: no global stack, no remote memory, and at most one in-flight child per pipe at a time. The recursion depth bound lives entirely in the `ray_meta_t.depth` field (3 bits, so `MAX_DEPTH` ≤ 7); the accumulator's 16-bit channel width plus the `total_shift`< 8 gate together bound the worst-case dynamic range, so the design never has to clip overflow inside the pipeline — only at the final clamp01 in stage H.

4.4 Resource Budget on the Target Device

The Cyclone V 5CSEMA5F31C6 has 32,070 ALMs, 87 variable-precision DSP blocks, and 397 M10K blocks. Table 3 shows the actual post-fit utilization for the delivered design at 125 MHz `divclk` and 100 MHz HPS bridge clock.

Resource	Used	Available	Utilization
ALMs	27,397	32,070	85%
Registers	56,313	128,280	44%
DSP blocks	60	87	69%
M10K blocks	22	397	5.5%
Block memory bits	194,560	4,065,280	4.8%
PLLs	1	6	17%
I/O pins	362	457	79%

Table 3: Quartus Fitter results for the deployed bitstream. ALM and DSP usage are the two binding constraints; the design has substantial M10K headroom because color BRAM is only one row deep, not one frame deep.

DSP usage is dominated by the four parallel pipes. Each pipe's two-sphere-and-one-cone intersection prep, hit-point reconstruction, normal recovery, reflection, light-direction, shadow, and shading stages together synthesize to roughly twelve to fifteen DSPs per pipe; the remaining DSPs are in the shared `basis_compute` and the time-muxed `sun_threshold_compute`. Within each stage, a single `fp_mul` is often time-muxed across several products by switching its operand mux on the per-stage tick counter — this is what lets each stage perform many multiplications inside the 43-cycle initiation interval without spending a DSP block per product. M10K usage is small because both of the design's pixel-scale memories are row-sized rather than frame-sized: the two-bank ping-pong color memory is $2 \times 480 \times 24 \approx 23$ kbit, and the shared 49-bit-per-tag reflection accumulator is $480 \times 49 \approx 23$ kbit. The frame buffer itself lives in HPS DDR3, not in M10K.

4.5 Memory System and DMA

A row contains 480 pixels. The color memory therefore needs only one row per bank:

$$2 \cdot 480 \cdot 24 = 23040 \text{ bits} \approx 2.8 \text{ KiB}.$$

This fits comfortably in M10K blocks and is much smaller than a full frame. Each row DMA uses 64-bit writes, packing two BGRX pixels per beat. A 480-pixel row therefore takes 240 accepted beats. The address for row y, starting column x, and frame slot s is

$$base + s \cdot (W \cdot H \cdot 4) + (yW + x) \cdot 4.$$

The HPS-side frame ring contains two full slots:

$$2 \cdot 480 \cdot 360 \cdot 4 = 1{,}382{,}400 \text{ bytes}.$$

The driver allocates this memory, maps it for DMA, writes its bus address into `FRAME_BUF_BASE`, and exposes a cached userspace mapping. The cache is safe because the driver invalidates the just-finished slot before userspace reads it.

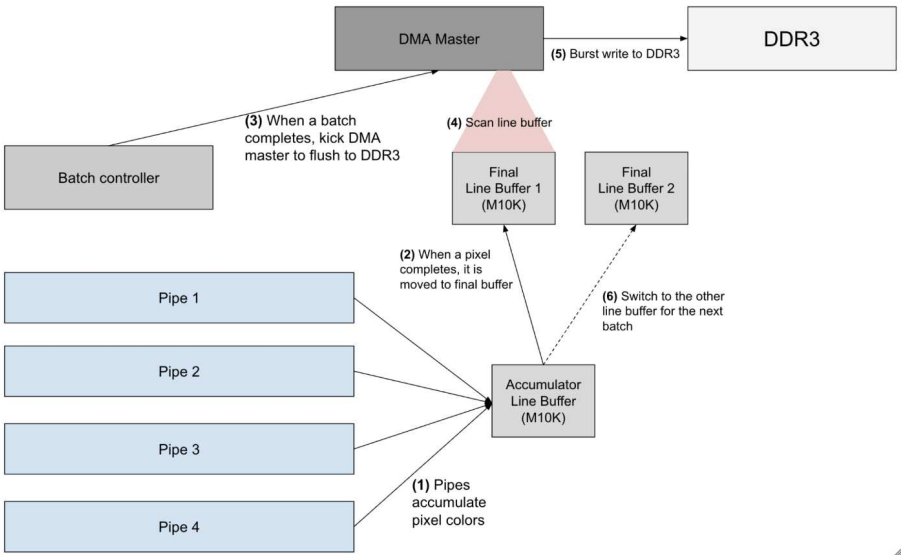


Figure 5: Pixel data path from the compute pipes to DDR3. Numbered steps: (1) the $K = 4$ pipes accumulate per-pixel color contributions into a shared accumulator BRAM, keyed by lane tag so reflection-chain bounces all land in the same slot. (2) When a chain terminates, the saturated 8-bit-per-channel result is written into the current line buffer. (3) When a batch (= one row) completes, the batch controller asserts `dma_kick` to the DMA master. (4) The DMA master streams the line buffer out, two pixels per 64-bit beat. (5) The Avalon-MM master burst writes 240 beats (1920 bytes) into the next row of the HPS DDR3 frame slot via the f2h_axi_slave bridge. (6) The line buffers ping-pong on `bank_sel`: the compute pipes write one bank while the DMA reads the other.

4.6 Frame FSM

The frame FSM has five states: `FRAME_IDLE`, `FRAME_KICK`, `FRAME_WAIT`, `FRAME_DRAIN`, and `FRAME_DONE_PULSE`. The important implementation choice is that `FRAME_WAIT` advances on `core_done`, not on DMA completion. This lets batch $N+1$ compute while batch N drains to DDR3. Only after the last row is kicked does `FRAME_DRAIN` wait for DMA completion and assert `FRAME_DONE`.

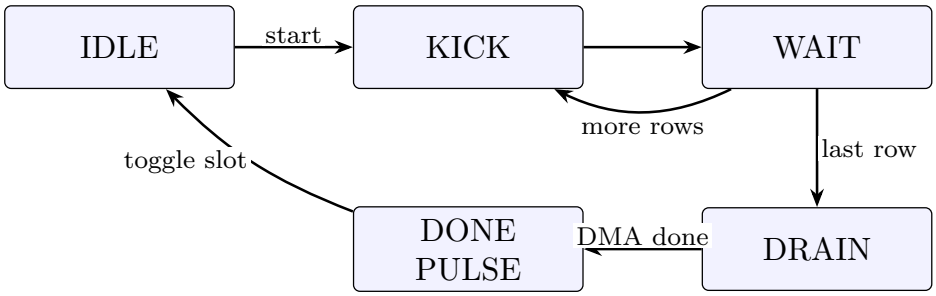


Figure 6: Frame FSM. Software starts a frame once; the hardware emits all row kicks internally.

5 Hardware/Software Interface

5.1 CSR Map

All signed scene quantities are Q14.13 unless noted. The CSR aperture is word-addressed by the Avalon slave and byte-addressed from software.

Offset	Register	Meaning
0x00	`CONTROL`	Bit 2 starts a frame. Bit 3 clears `FRAME_DONE`.
0x04	`STATUS`	Bit 2 `FRAME_DONE`, bit 3 `FRAME_RUNNING`, bit 4 current `WRITE_SLOT`.
0x10–0x18	`LIGHT_X/Y/Z`	Light vector and sun direction input.
0x1c–0x20	`R0_SQ`, `R1_SQ`	Squared sphere radii, precomputed by software.
0x24	`DMA_CTRL`	Driver-owned DMA enable/clear bits.
0x28	`DMA_STATUS`	DMA busy/error/done and debug state.
0x3c	`FRAME_BUF_BASE`	Driver-owned physical base address of the two-slot DDR3 frame ring.
0x40–0x48	`SC1_X/Y/Z`	Sphere 1 center.
0x4c	`MAX_DEPTH`	3-bit reflection recursion cap. Zero disables reflection.
0x50–0x54	`REFLECT_SHIFT_0/1`	Per-sphere power-of-two attenuation shifts.
0x58–0x5c	`CAM_X/Z`	Camera position. Camera y is fixed in hardware at 1.5.
0x60–0x7c	`RIGHT`, `UP`, `FWD`	Camera basis vector components. `right.y` is hardcoded as zero.
0x80–0x88	`SC0_X/Y/Z`	Sphere 0 center.
0x8c–0xa0	`PC1/PC2_RGB`	Two floor colors.
0xa4–0xb8	`SCOL0/SCOL1_RGB`	Two sphere colors.
0xbc	`FLOOR_MODE`	Checker, Sierpinski, stripe, or fine-checker floor pattern.
0xc8–0xec	Cone CSRs	Apex, k^2, height, color, reflection shift, and normal factor.
0x100+	`COLOR[]`	Debug color-memory mirror; production readback uses DMA.

5.2 Driver API

The kernel module registers `/dev/raytracer`. Userspace first calls `RTB_IOCTL_GET_LAYOUT`, then maps two regions:

- offset 0: the uncached CSR aperture, cast to `volatile struct rtb_csrs *`;
- offset 4096: the cached two-slot frame ring, read-only in userspace.

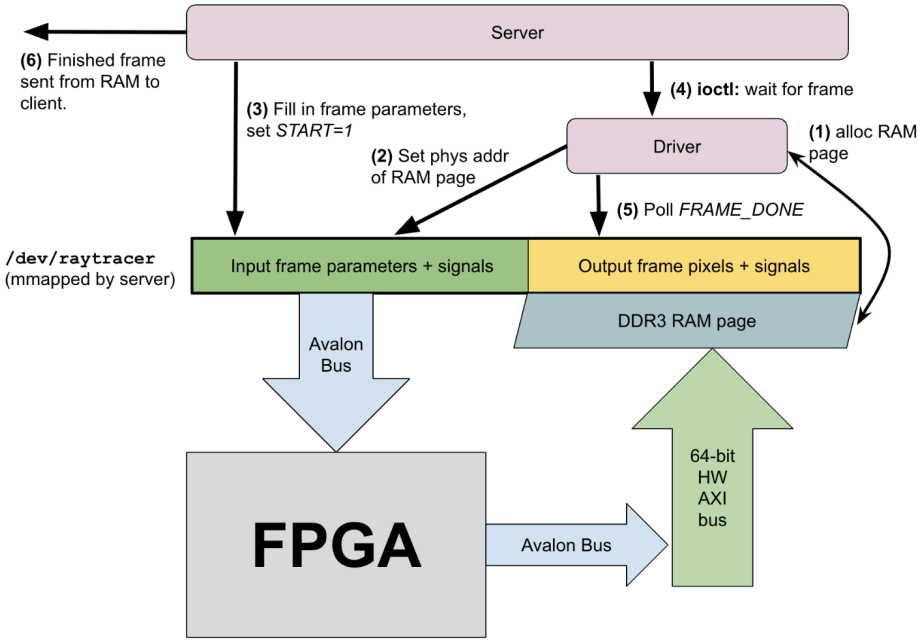


Figure 7: Per-frame handshake between the server, the driver, and the FPGA via the `/dev/raytracer` mmap interface. (1) At probe time the driver allocates a contiguous DDR3 page and registers it for DMA. (2) The driver writes its bus address into the FPGA's `FRAME_BUF_BASE` CSR (driver-owned, never touched by userspace). (3) The server mmaps the CSR region, fills in the input-frame parameters, and writes `CONTROL[2]` to start a frame. The CSR writes traverse the Avalon bus into the FPGA. (4) The server issues `RTB_IOCTL_WAIT_FRAME`; (5) the driver polls `FRAME_DONE` and, when set, invalidates the just-finished slot's CPU caches and returns the slot index. The FPGA wrote the slot through the heavyweight AXI master in parallel. (6) The server reads the cached pixel mmap and sends the frame to the client.

For each frame, userspace writes all scene CSRs and then writes `RTB_CONTROL_FRAME_START`. It waits with `RTB_IOCTL_WAIT_FRAME`, which spin-polls the hardware status with a timeout, clears the done sticky bit, invalidates the CPU cache lines for the finished frame slot, and returns the finished slot index.

5.3 Network Wire Protocol

The desktop-to-HPS packet is 49 big-endian 32-bit words, or 196 bytes. It contains light position, sphere and cone parameters, reflection controls, camera position, camera basis, floor colors, sphere colors, floor mode, floor bounds, and the cone normal factor. The HPS-to-desktop payload is one raw BGRX frame: $480 \times 360 \times 4 = 691{,}200$ bytes. The desktop uses PIL's raw `BGRX` decoder to avoid a Python per-pixel conversion loop.

5.4 File and Wire Formats

The system manipulates four distinct serialized formats, summarized below.

Per-frame input packet (client → HPS). 49 signed big-endian `int32` words, 196 bytes total. Word indices and contents are: `[0..2]` light $\{x, y, z\}$; `[3..5]` sphere-1 center; `[6]` `max_depth` (0–7); `[7..8]` per-sphere reflectivity shifts; `[9..10]` camera $\{x, z\}$; `[11..12]` basis `right.{x,z}`; `[13..15]` basis `up.{x,y,z}`; `[16..18]` basis `fwd.{x,y,z}`; `[19..20]` r_0^2, r_1^2; `[21..23]` sphere-0 center; `[24..29]` floor colors PC1/PC2; `[30..35]` sphere colors SCOL0/SCOL1; `[36]` `floor_mode`; `[37..38]` floor half-extents; `[39..41]` cone apex; `[42]` k^2; `[43]` cone height; `[44..46]` cone color; `[47]` cone reflectivity shift; `[48]` `cone_norm_factor` $= \sqrt{k^2(1+k^2)}$. All Q-typed fields are in Q14.13.

Frame buffer (HW → DDR3 → HPS userspace → socket). Each frame slot is a contiguous $480 \times 360 \times 4 = 691{,}200$-byte BGRX region, row-major, top-to-bottom. The byte order per pixel is `B, G, R, 0x00`. The frame ring has two such slots back-to-back (1,382,400 bytes total). The HPS reads from the just-finished slot after `RTB_IOCTL_WAIT_FRAME` has cache-invalidated it.

PPM output (optional debug, `render_frame.c`). A single-frame command-line tool emits a 480×360 binary `P6` PPM. The file is a 15-byte ASCII header (`P6\n480 360\n255\n`) followed by $480 \times 360 \times 3 = 518{,}400$ bytes of packed $\{R, G, B\}$, in the same row-major order as the frame buffer with the alpha byte stripped.

Joystick events. The desktop client reads SDL joystick events through pygame in `SW/desktop/joystick.py`. Two axes are consumed: `axis[0]` (left/right) drives yaw and `axis[1]` (up/down) drives pitch when held in the analog deadzone $|axis| > 0.2$. A trigger button switches the joystick from camera control to light control in the `joystick_param_game.py` demo. No file is written; events are consumed at frame rate and folded into the next outgoing input packet.

6 Software Design

6.1 Kernel Driver

The driver is a small platform/misc driver matched by the device-tree compatible string `csee4840,raytracer-1.0`. During probe it registers the misc device, maps the hardware CSR resource, allocates the two-slot frame buffer with `alloc_pages_exact`, maps it for DMA with `dma_map_single`, checks that the bus address fits in the 32-bit hardware register, writes `FRAME_BUF_BASE`, and enables the DMA master. The driver owns DMA setup and cache coherency; userspace owns scene parameters.

6.2 HPS Server

The HPS server owns the device file and listens for one TCP client at a time. Its steady-state loop is deliberately ordered as `WAIT_FRAME`, `kick_frame(next)`, `send_all(done)`, `recv_frame_inputs(nextnext)`. Starting the next frame before sending the finished frame lets the FPGA work while Linux copies the previous frame into the socket buffer.

6.3 Desktop Client and Demos

The desktop client uses pygame for display and optional joystick input. It computes the camera basis on the desktop, converts floats to Q14.13 integers, sends the 196-byte input packet, receives a BGRX

frame, and displays it. Additional scripts build demos on top of the same protocol, including camera movement, cone scenes, a garden/sun scene, and a Pong-style demo where spheres and the cone are animated as game objects.

7 Verification and Results

7.1 Verification Layers

The repository contains three layers of verification. First, readable Python reference renderers describe the intended algorithm in floating point and fixed point. Second, cocotb tests exercise the fixed-point multiplier, divider, inverse, square root, cone math, and the batch raytracing pipeline against Python reference calculations. Third, the HPS tools provide an end-to-end hardware path: the kernel driver can render one frame to a PPM through `render_frame.c`, and the TCP server/client pair can stream interactive frames.

The most important functional tests are the pixel-level comparisons in `test_raytracer_batch.py`. These cover default scenes, reflections, configurable colors, floor modes, and cone behavior. The tests allow small fixed-point tolerances near edges, where a one-LSB difference in an intersection can select a neighboring branch.

7.2 Performance Evolution

Three architectural decisions account for most of the realized speedup, each landing as a measurable jump in frame rate. Table 5 summarizes the progression at the same scene complexity (two spheres on the default floor, no reflections, 480×360).

Configuration	Pipelines	Per-frame	Frame rate	Dominant cost
Unpipelined SV on FPGA	None	~ 500 ms	~ 2 fps	Direct port of reference algorithm; one ray at a time, no DMA.
HW + per-batch `ioctl`	3	~ 90 ms	~ 11 fps	$\sim 50\,\mu$s SW per batch $\times$ 1440 batches/frame.
HW + mmap, per-batch SW poll	3	~ 50 ms	~ 20 fps	Userspace polling + per-batch memcpy.
HW frame engine (one CSR/frame)	4	~ 21 ms	~ 48 fps	Compute-bound at 100 MHz `divclk`.
+ compute/DMA overlap + 125 MHz (≤ 7 reflections)	4	~ 17 ms	~ 60 fps	DMA tail + cache invalidate + TCP send.
HW Q14.10, $K = 6$; one sphere, no cone, no reflections	6	~ 7 ms	~ 140 fps	I/O-bound (DMA, TCP)

Table 5: Performance evolution. The top five rows are chronological checkpoints in the same scene (two spheres + cone + floor, 480×360); the biggest single jump is the move from a per-batch `ioctl` kick to an HW-driven autonomous frame engine that overlaps batch $N + 1$'s compute with batch N's DMA, dropping per-frame SW overhead from tens of milliseconds to a single ioctl wait. The final 125 MHz clock-up adds another $\sim 16\%$ on top. The last row (below the rule) is an intermediate-revision data point at narrower word width and simpler geometry, showing what the I/O path is capable of when the compute pipeline is not the bottleneck.

The baseline is the reference algorithm transliterated directly into SystemVerilog — one ray at a time, no pipelining, no parallelism, no DMA — and synthesized for the same FPGA. It is correct but uninteractive: roughly half a second per frame, because the design serializes every primitive intersection and shading step rather than overlapping them. Adding a four-stage pipeline and moving the per-pixel math through Avalon-MM batches but keeping the per-batch software synchronization gets the system to ~ 11 fps; the bottleneck is then the $\sim 60\,\mu$s round-trip per `ioctl` call, not the FPGA. Switching the kernel-userspace boundary from copy ioctls to `mmap`'ed CSR and DMA regions removes the copy cost but leaves the per-batch synchronization, so the system reaches ~ 20 fps. The decisive jump is the hardware frame engine: after Stage 3 the only software work per frame is one `FRAME_START` CSR write and one `WAIT_FRAME` ioctl, the HW chains 360 batches autonomously, and the DMA master overlaps with the next batch's compute through the two-bank color BRAM. At that point the system is hardware-bound, and Stage 4's double-buffered frame ring plus the move from 100 MHz to 125 MHz `divclk` produce the final result.

7.2.1 Scaling with Scene Complexity and Resolution

The numbers in Table 5 are for the default scene at 480×360 with reflections disabled. Two natural axes affect the steady-state frame rate, and both behave the way one would expect for a compute-bound design:

- **Scene complexity (reflections).** Enabling reflections costs additional ray issues but does *not* cost any additional primary rays. A bouncing chain through one sphere costs roughly one additional ray issue per reflective hit, and the spawn-preference arbiter prevents reflection rays from being starved. With `MAX_DEPTH` $= 3$ and the default 25%/50% sphere reflectivities the measured frame rate drops by about 5–8 fps relative to the no-reflection baseline (from ~ 58 to $\sim$ 50–53 fps), and with both reflectivities disabled the system returns to the upper bound. The cone path costs roughly the same per-bounce as the sphere path because it uses the same quadratic intersection and the same accumulator structure.
- **Resolution.** Lowering the render resolution below 480×360 scales the frame time roughly linearly with pixel count, because compute dominates and the DMA tail and TCP overhead are amortized over the whole frame. A trimmed-scene preview at smaller resolutions (used during bring-up to keep wall-clock per simulation rerun low) crossed the 120 fps mark; that number is reported for completeness rather than as a target. It is interesting only because it confirms that the row-DMA path is not the bottleneck even at much shorter per-row times, and that the system would scale to a higher frame rate if the resolution could be lowered in production.

The interactive demos in `SW/desktop/` (camera fly-through, garden scene, Pong-style ball game) all run at the steady-state ~ 58 fps with reflections on, with no perceptible jitter from the joystick path. The remaining wall-clock budget — the few-millisecond gap between the ~ 14.9 ms compute lower bound and the measured ~ 17 ms per-frame time — is split between the DMA drain on the last row, the L2 cache invalidate inside the WAIT_FRAME ioctl, and the 691 KB TCP `send` into the HPS-to-desktop socket. None of these are dominant on their own.

Earlier-configuration data point. For comparison, an intermediate revision of the design — 24-bit Q14.10 fixed-point, $K = 6$ lanes, simplified scene geometry (one sphere on the plane, no cone, reflections off) — reached roughly 7 ms per frame, at which point the system was I/O-bound rather than compute-bound (DMA tail plus TCP send dominated, not the pipeline). We did not preserve that configuration

in the final design because the wider word and the extra primitives mattered more than the headline frame-rate number, but it confirms that the row-DMA path and the network stack have substantial headroom above the current 58 fps operating point.

8 Division of Work, Lessons, and Advice

8.1 Division of Work

- Innokentiy Kaurov worked on algorithm design, the client program, and the server.
- Matthew Lou worked on hardware optimizations, pipeline design, and kernel drivers.
- Tony Giannini worked on algorithm design and the hardware/software interface.

8.2 Lessons Learned

The largest lesson is that the hard part of an FPGA raytracer is not only the ray math. It is scheduling the math so that fixed-latency units, memory ports, bus bursts, and software synchronization all line up. The renderer was not bottlenecked by one single hard problem; it was bottlenecked by the alignment of many medium-sized problems — arithmetic latency, bus granularity, cache coherency, routing delay, software synchronization, and verification. Moving camera trigonometry and cone normal precomputation into software was a good trade because those values change once per frame, while pixel math repeats 172,800 times per frame. The row-DMA design was equally decisive: it avoids per-pixel software intervention and uses only a small amount of on-chip memory.

A related lesson is that the right place for a full frame is DDR3, not on-chip memory or a wide register aperture. A row-sized M10K buffer is excellent as a staging structure — it sits next to the compute core and feeds the DMA master at full bus width — but a 691 KB frame does not belong in fabric memory. Once the DDR3 path was in place, cache coherency stopped being a cleanup detail and became part of the public API: because Linux reads the just-finished frame slot cached, the driver has to invalidate the slot's L1/L2 lines before returning from `RTB_IOCTL_WAIT_FRAME`. Folding the invalidate into the wait ioctl made userspace incapable of accidentally reading stale data.

Timing closure dominated the final phase. Wide scene parameters broadcast to every pipeline stage create long routes, so the design uses per-pipe relay registers for batch-stable signals — these were not cosmetic, they were part of timing closure. Single-cycle multipliers are registered to use DSP output flip-flops. The cone path exposed how easy it is for a mathematically small change to arrive one tick too late in a fixed-II pipeline; moving the cone a precomputation earlier restored a full stage of timing budget. The 41-cycle divider in particular sets the rhythm of the whole design: not just the stages that divide, but the initiation interval, lane staggering, accumulator schedule, and spawn timing. Once the pipeline is fixed-II, every new feature has to land on the right tick.

8.3 Advice for Future Projects

Start with a simple reference renderer and keep it runnable throughout the project — the cocotb pixel-level tests are only as useful as the reference model behind them, and the Python floating-point and fixed-point renderers were what made it possible to debug cone intersections, reflection edge cases, floor modes, and shadow tests without guessing from screenshots. Decide early which quantities are per-pixel and which are per-frame; anything constant for all 172,800 pixels of a frame should be considered

for software precomputation before adding another divider, square root, or trigonometric datapath in hardware.

Before iterating on tactical fixes, take a step back and sketch the high-level pipeline. Counting cycles end-to-end on paper — how long a single ray takes through every stage versus how often a new ray can be issued — exposes whether the design is actually pipelined or just sequential, and that judgement should drive the next change. We learned this the slow way: early on we kept tightening individual stages and adding lanes for incremental gains, but the real ceiling was that each ray had to walk every stage to completion before the next ray could enter. Moving to a fixed-II pipelined architecture, where stages run concurrently on rays at different points in flight, unlocked an order-of-magnitude improvement that no amount of stage-level tuning could have produced. Brute-force iteration on a misidentified bottleneck buys very little, and the cost of being wrong is high in an FPGA project where every turn of the loop is a ten-minute build.

Finally, do not wait until the end to run timing. A design can pass simulation for weeks and still fail because one broadcast net or one mux arrives half a nanosecond too late.

8.4 Use of AI Assistants

We used Claude (Anthropic) as a coding assistant throughout the project. Its most useful contributions were narrow and verifiable: drafting and editing cocotb test harnesses against the Python reference renderer, iterating on timing-closure fixes, and producing first drafts of the LaTeX figures and tables in this report. AI was also used to annotate the SystemVerilog and C header sources — adding short one-line port-purpose comments to every module port and writing the human-readable schedule comments at the top of each pipeline stage.

The single most productive use of AI on the project was timing-closure iteration, and once the loop was reliable we ran it autonomously overnight. The build loop is long — a full Quartus compile plus STA takes roughly ten minutes on the lab build server — so closing the 125 MHz target involved running through dozens of RTL revisions over several days. Each iteration looked the same: the agent was given the most recent `rt_divclk_paths.rpt`, asked to identify which path was now critical, and asked to propose a structural fix (defer a dot-sum by one tick, pre-register a slab subtract, migrate a multiply to the DSP output FF, hoist a precomputation into an earlier stage). Because each cycle was build-bound rather than reasoning-bound, batching the iterations into an unattended overnight run was a much better use of wall-clock time than babysitting them interactively — the agent would propose a fix, push, wait for the remote harness to compile and report new slack, and either commit or roll back based on the result, then pick the next critical path and repeat.

The reason this was safe was the cocotb pixel-level test suite. Every proposed RTL change first ran against the fixed-point Python reference renderer for the default scene, the reflection scene, the cone scene, and the floor-mode variants. A timing fix that silently broke pixel correctness was caught in seconds rather than discovered hours later in a hardware demo, so we could let the agent propose moderately aggressive register shuffles — and run that loop unattended — without losing trust in the design. Without the test harness this loop would have been actively dangerous: the agent is good at pattern-matching "DSP → 3-input add → register" anti-patterns from an STA report, but it has no independent way to tell whether the resulting RTL still computes the right value.

The AI was *not* used as a substitute for design judgement. The architectural decisions in this report — moving camera trigonometry and cone normal precomputation into software, choosing $K = 4$ staggered lanes over a deeper or wider configuration, double-banking the color BRAM so compute and DMA overlap, and the side-channel `ray_meta_t` for reflection bookkeeping — were human choices made on

the strength of profiling data and synthesis reports. We mention this section explicitly because the course encourages transparency about tool use, but we have intentionally kept the boundary clean: AI accelerated implementation and documentation work, the design itself is ours.

References

[1] Stephen A. Edwards, *CSEE 4840 Embedded System Design, Spring 2026*. https://www.cs.columbia.edu/~sedwards/classes/2026/4840-spring/index.html

[2] Terasic Technologies, *DE1-SoC User Manual (rev.F/rev.G Board)*, Version 2.0.4, Jan. 2019. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4

[3] Turner Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, June 1980. https://doi.org/10.1145/358876.358882

[4] Peter Shirley, Trevor David Black, and Steve Hollasch, *Ray Tracing in One Weekend*, Version 4.0.2, Apr. 2025. https://raytracing.github.io/books/RayTracingInOneWeekend.html

[5] Behrooz Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed., Oxford University Press, 2010.

[6] Miloš D. Ercegovac and Tomás Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.

A Module Index and Source Listing Policy

The following appendix lists the authored source files most relevant to building, testing, and demonstrating the project.

A.1 Verilog Modules

All RTL lives in a single SystemVerilog file (`raytracer.sv`) plus a small package (`raytracer_pkg`). Table 6 enumerates the modules, their role, and where they are instantiated.

Table 6: Verilog modules in `raytracer.sv`. Most stage modules share the same parameter set: `FRAC_BITS=13`, `WORD=27`, `TAG_W=7`; `rtl_batch_pipe` additionally takes `K=4` and `BATCH=480`.

Module	Role	Instantiated in
`raytracer_batch_mm`	synthesis top, CSR + DMA host	`soc_system.qsys`
`rtl_batch_pipe`	K-lane compute core	`raytracer_batch_mm`
`rtl_pipe`	single per-lane fixed-II pipeline	`rtl_batch_pipe` ($K = 4$)
`rtl_ray_gen`	stage A: ray gen + $\|d\|^2$	`rtl_pipe`
`rtl_isect_prep`	stage B: sphere/-cone quadratic prep	`rtl_pipe`
`rtl_hit_resolve`	stage C: nearest-hit + hit point	`rtl_pipe`
`rtl_normal_div`	stage D: 1/‖precursor‖	`rtl_pipe`
`rtl_reflect_dir`	stage E: normal + reflection dir	`rtl_pipe`
`rtl_light_dir_div`	stage F: $1/\|L - p\|$	`rtl_pipe`
`rtl_shadow_test`	stage G: shadow ray + occluder check	`rtl_pipe`
`rtl_shade_blend`	stage H: Lambert + sky/sun + clamp	`rtl_pipe`
`basis_compute`	per-frame camera basis (DSP)	`rtl_batch_pipe`
`rtl_dma_master`	Avalon-MM master / 240-beat burst	`raytracer_batch_mm`
`sun_threshold_compute`	$(63/64)\|L\|^2$ time-muxed	`raytracer_batch_mm`
`fp_mul`	registered 27×27 multiply	every stage with a variable multiply

Module	Role	Instantiated in
`fp_sqrt`	bit-serial sqrt (∼20 c)	`rtl_isect_prep`, `rtl_hit_resolve`, `rtl_reflect_dir`, `rtl_shadow_test`
`fp_inv`	bit-serial reciprocal (∼41 c)	`rtl_isect_prep`, `rtl_normal_div`, `rtl_light_dir_div`
`fp_div`	bit-serial divider	instantiated only inside `fp_inv`

A.2 Authored Source Files

- `raytracer.sv` — all RTL (∼4000 LOC).
- `raytracer_hw.tcl`, `soc_system.qsys`, `soc_system.tcl`, `soc_system.sdc` — Qsys/Quartus integration, timing constraints.
- `SW/raytracer_batch.h` — shared C header (CSR mirror, ioctl numbers, mmap layout).
- `SW/raytrace_batch_drve.c` — Linux kernel driver.
- `SW/fpga/server.c`, `SW/fpga/render_frame.c` — HPS userspace.
- `SW/desktop/client.py`, `camera.py`, `joystick.py`, plus the demo scripts under `SW/desktop/` — desktop client and demos.
- `raytracer_fp.py`, `human_raytracer.py` — Python reference renderers (fixed and float).
- `test_raytracer_batch.py`, `test_fp*.py` — cocotb tests.

CS4840 Embedded Systems

FPGA Raytracer — Source Code

May 13, 2026

Contents

1 Hardware (SystemVerilog + Qsys + Quartus)

raytracer.sv

```
// ============================================================================
// Package: per-ray reflection metadata.
//
// `ray_meta_t` is an 11-bit packed struct that travels alongside each ray
// through every pipeline stage. For primary rays it's all zero; reflection
// rays inherit/derive their values when the parent's stage G spawns a child:
//
//   depth        -- bounce count. 0 = primary; incremented on each spawn.
//                   Stage G refuses to spawn when depth >= max_depth.
//   excluded     -- one-hot bitmask of which primitive this ray was just
//                   spawned off of. Stage C masks that primitive's hit so
//                   the new ray doesn't immediately self-intersect at the
//                   surface origin. excluded[0]=skip s0, [1]=skip s1,
//                   [2]=skip cone. 000=none (primary).
//   total_shift  -- cumulative power-of-2 attenuation along the chain. Each
//                   spawn adds the parent-primitive's reflect_shift. The
//                   accumulator emits (color × bright) >> total_shift, so
//                   deep bounces contribute progressively less.
// ============================================================================
package raytracer_pkg;
    localparam int RT_WORD  = 27;
    // Tag width sized for production BATCH=480.
    localparam int RT_TAG_W = 9;

    typedef struct packed {
        logic [2:0] depth;        // 0 (primary) .. max_depth (terminal)
        logic [2:0] excluded;     // {cone, s1, s0} one-hot self-skip bitmask
        logic [4:0] total_shift;  // running >> count for accumulator weight
    } ray_meta_t;

    typedef struct packed {
        logic signed [RT_WORD-1:0] r, g, b;
    } rgb_t;

    // Sphere primitive - geometry + surface appearance + reflectivity.
    // r_sq is pre-squared by SW.
    typedef struct packed {
        logic signed [RT_WORD-1:0] x, y, z;
        logic signed [RT_WORD-1:0] r_sq;
        rgb_t                      color;
        logic [4:0]                reflect_shift;
    } sphere_t;

    // Y-axis cone primitive. Apex on top at (apex_x, apex_y, apex_z),
    // single-half (opens downward) over y   [apex_y - height, apex_y].
    // Surface (-xAx)² + (-zAz)² = k_sq · (-yAy)². Bottom is capped by a
    // disc of radius² = k_sq · height². norm_factor = sqrt(k_sq · (1+k_sq))
    // is SW-precomputed to skip a sqrt in HW. reflect_shift = 0 → matte.
    typedef struct packed {
        logic signed [RT_WORD-1:0] apex_x, apex_y, apex_z;
        logic signed [RT_WORD-1:0] k_sq;
        logic signed [RT_WORD-1:0] height;
        logic signed [RT_WORD-1:0] norm_factor;
        rgb_t                      color;
        logic [4:0]                reflect_shift;
    } cone_t;

    // Floor (infinite-ish ground plane) - checker colors + pattern mode +
    // bounded half-extents. Consumed only by rtl_hit_resolve.
    typedef struct packed {
        rgb_t                      pc1, pc2;
        logic [1:0]                mode;
        logic signed [RT_WORD-1:0] lim_x, lim_z;
    } floor_t;

    // Top-level scene bundle. Each stage receives this and reads only the
    // fields it needs; Quartus prunes unused field-driven logic.
    typedef struct packed {
        sphere_t s0, s1;
```

```
        cone_t   cone;
        floor_t  floor;
    } scene_t;

    // Per-ray ride-along through every stage handoff.
    typedef struct packed {
        logic                  sv;
        logic [RT_TAG_W-1:0]   tag;
        ray_meta_t             meta;
    } pipe_baggage_t;

    localparam logic [2:0] HIT_TYPE_SKY   = 3'b000;
    localparam logic [2:0] HIT_TYPE_S0    = 3'b001;
    localparam logic [2:0] HIT_TYPE_S1    = 3'b010;
    localparam logic [2:0] HIT_TYPE_PLANE = 3'b011;
    localparam logic [2:0] HIT_TYPE_CONE  = 3'b100;

    // Floor pattern modes; selects rtl_hit_resolve's chk_bit derivation.
    localparam logic [1:0] FLOOR_MODE_1M_CHECK   = 2'b00;
    localparam logic [1:0] FLOOR_MODE_SIERPINSKI = 2'b01;
    localparam logic [1:0] FLOOR_MODE_DIAGONAL   = 2'b10;
    localparam logic [1:0] FLOOR_MODE_FINE_CHECK = 2'b11;

    localparam int ADDR_CONTROL              = 7'd0;
    localparam int ADDR_STATUS               = 7'd1;
    localparam int ADDR_LIGHT_X         = 7'd4;
    localparam int ADDR_LIGHT_Y         = 7'd5;
    localparam int ADDR_LIGHT_Z         = 7'd6;
    localparam int ADDR_R0_SQ           = 7'd7;
    localparam int ADDR_R1_SQ           = 7'd8;
    localparam int ADDR_DMA_CTRL             = 7'd9;
    localparam int ADDR_DMA_STATUS           = 7'd10;
    localparam int ADDR_FRAME_BUF_BASE       = 7'd15;
    localparam int ADDR_SC1_X           = 7'd16;
    localparam int ADDR_SC1_Y           = 7'd17;
    localparam int ADDR_SC1_Z           = 7'd18;
    localparam int ADDR_MAX_DEPTH       = 7'd19;
    localparam int ADDR_REFLECT_SHIFT_0 = 7'd20;
    localparam int ADDR_REFLECT_SHIFT_1 = 7'd21;
    localparam int ADDR_CAM_X           = 7'd22;
    localparam int ADDR_CAM_Z           = 7'd23;
    localparam int ADDR_RIGHT_X         = 7'd24;
    localparam int ADDR_RIGHT_Z         = 7'd25;
    localparam int ADDR_UP_X            = 7'd26;
    localparam int ADDR_UP_Y            = 7'd27;
    localparam int ADDR_UP_Z            = 7'd28;
    localparam int ADDR_FWD_X           = 7'd29;
    localparam int ADDR_FWD_Y           = 7'd30;
    localparam int ADDR_FWD_Z           = 7'd31;
    // Visual-upgrades CSRs -(#0#5)
    localparam int ADDR_SC0_X           = 7'd32;
    localparam int ADDR_SC0_Y           = 7'd33;
    localparam int ADDR_SC0_Z           = 7'd34;
    localparam int ADDR_PC1_R           = 7'd35;
    localparam int ADDR_PC1_G           = 7'd36;
    localparam int ADDR_PC1_B           = 7'd37;
    localparam int ADDR_PC2_R           = 7'd38;
    localparam int ADDR_PC2_G           = 7'd39;
    localparam int ADDR_PC2_B           = 7'd40;
    localparam int ADDR_SCOL0_R         = 7'd41;
    localparam int ADDR_SCOL0_G         = 7'd42;
    localparam int ADDR_SCOL0_B         = 7'd43;
    localparam int ADDR_SCOL1_R         = 7'd44;
    localparam int ADDR_SCOL1_G         = 7'd45;
    localparam int ADDR_SCOL1_B         = 7'd46;
    localparam int ADDR_FLOOR_MODE      = 7'd47;
    localparam int ADDR_FLOOR_LIM_X     = 7'd48;
    localparam int ADDR_FLOOR_LIM_Z     = 7'd49;
    localparam int ADDR_CONE_X               = 7'd50;
    localparam int ADDR_CONE_Y               = 7'd51;
    localparam int ADDR_CONE_Z               = 7'd52;
    localparam int ADDR_CONE_K_SQ            = 7'd53;
    localparam int ADDR_CONE_HEIGHT          = 7'd54;
    localparam int ADDR_CONE_COL_R           = 7'd55;
```

```
    localparam int ADDR_CONE_COL_G          = 7'd56;
    localparam int ADDR_CONE_COL_B          = 7'd57;
    localparam int ADDR_REFLECT_SHIFT_CONE  = 7'd58;
    localparam int ADDR_CONE_NORM_FACTOR    = 7'd59;
    localparam int ADDR_COLOR_0             = 7'd64;
endpackage : raytracer_pkg

// fp_mul: (a*b) >> FRAC_BITS. Output registered (Quartus infers the DSP's
// hard output FF). Latency: 1 cycle.
module fp_mul #(
    parameter FRAC_BITS = 13,
    parameter WORD      = 27
)(
    input  logic                    clk,    // clock; output registered on posedge
    input  logic signed [WORD-1:0] a,      // multiplicand (Q14.13)
    input  logic signed [WORD-1:0] b,      // multiplier   (Q14.13)
    output logic signed [WORD-1:0] result  // a·b, registered, 1-cycle latency
);
    logic signed [2*WORD-1:0] full;
    assign full = a * b;
    always_ff @(posedge clk) begin
        result <= full[WORD-1+FRAC_BITS:FRAC_BITS];
    end
endmodule

// Asymmetric-width fp_mul: lets basis_compute use 14-bit px_c against
// 27-bit basis vectors so Quartus infers a 14×27 (1 DSP) instead of 27×27.
module fp_mul_narrow #(
    parameter int FRAC_BITS = 13,
    parameter int WA        = 27,
    parameter int WB        = 14,
    parameter int WOUT      = 27
)(
    input  logic                    clk,    // clock; output registered
    input  logic signed [WA-1:0]   a,      // wide operand (WA bits, Q14.FRAC_BITS)
    input  logic signed [WB-1:0]   b,      // narrow operand (WB bits, Q*.FRAC_BITS) - small range
    output logic signed [WOUT-1:0] result  // a·b in WOUT bits; 1-cycle latency
);
    logic signed [WA+WB-1:0] full;
    assign full = a * b;
    always_ff @(posedge clk) begin
        result <= full[WOUT-1+FRAC_BITS:FRAC_BITS];
    end
endmodule

// fp_sqrt: digit-by-digit. Latency = (WORD+FRAC_BITS)/2 cycles (20 at Q14.13).
// Reference: https://projectf.io/posts/square-root-in-verilog/
module fp_sqrt #(
    parameter FRAC_BITS = 13,
    parameter WORD      = 27
)(
    input  logic                    clk,    // clock
    input  logic                    rst_n,  // active-low reset
    input  logic                    start,  // 1-cycle pulse: latch a, begin sqrt
    input  logic signed [WORD-1:0] a,      // radicand (Q14.13); a<=0 → result=0
    output logic signed [WORD-1:0] result, // √a (Q14.13), valid when done=1
    output logic                    done    // 1-cycle pulse 21c after start (start latch + ITER=20 iterations)
);
    localparam ITER = (WORD + FRAC_BITS) >> 1;

    logic [WORD-1:0]       x, x_next;
    logic [WORD-1:0]       q, q_next;
    logic [WORD+1:0]       ac, ac_next;
    logic [WORD+1:0]       test_res;
    logic [$clog2(ITER):0] i;
    logic                  busy;

    always_comb begin
        test_res = ac - {q, 2'b01};
        if (test_res[WORD+1] == 0) begin
            {ac_next, x_next} = {test_res[WORD-1:0], x, 2'b0};
            q_next = {q[WORD-2:0], 1'b1};
        end else begin
            {ac_next, x_next} = {ac[WORD-1:0], x, 2'b0};
```

```
            q_next = q << 1;
        end
    end

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            result <= '0;
            done   <= 1'b0;
            busy   <= 1'b0;
            q      <= '0;
            ac     <= '0;
            x      <= '0;
            i      <= '0;
        end else begin
            done <= 1'b0;
            if (start) begin
                if (a <= 0) begin
                    result <= '0;
                    done   <= 1'b1;
                    busy   <= 1'b0;
                end else begin
                    busy <= 1'b1;
                    i    <= '0;
                    q    <= '0;
                    {ac, x} <= {{WORD{1'b0}}, a, 2'b0};
                end
            end else if (busy) begin
                if (i == ITER - 1) begin
                    busy   <= 1'b0;
                    done   <= 1'b1;
                    result <= q_next;
                end else begin
                    i  <= i + 1;
                    x  <= x_next;
                    ac <= ac_next;
                    q  <= q_next;
                end
            end
        end
    end
endmodule

// fp_div: (a << FRAC_BITS) / b. Sequential restoring divider, WIDE+1c
// latency. Returns 0 on b==0.
module fp_div #(
    parameter FRAC_BITS = 13,
    parameter WORD      = 27
)(
    input  logic                  clk,    // clock
    input  logic                  rst_n,  // active-low reset
    input  logic                  start,  // 1-cycle pulse: latch a/b, begin divide
    input  logic signed [WORD-1:0] a,     // dividend  (Q14.13)
    input  logic signed [WORD-1:0] b,     // divisor   (Q14.13); b==0 → result=0
    output logic signed [WORD-1:0] result,// a/b (Q14.13), valid when done=1
    output logic                  done    // 1-cycle pulse 41c after start
);
    localparam WIDE = WORD + FRAC_BITS;

    logic                 sign_out;
    logic [WIDE-1:0]      numer;
    logic [WIDE-1:0]      denom;
    logic [WIDE-1:0]      quotient;
    logic [WIDE-1:0]      remainder;
    logic [WIDE-1:0]      rem_next;
    logic [$clog2(WIDE):0] count;

    typedef enum logic [1:0] {
        IDLE,
        BUSY,
        FINISH
    } state_t;
    state_t state;

    always_comb begin
```

```
        rem_next = {remainder[WIDE-2:0], numer[count-1]};
    end

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            result   <= '0;
            done     <= 1'b0;
            state    <= IDLE;
            numer    <= '0;
            denom    <= '0;
            quotient <= '0;
            remainder<= '0;
            count    <= '0;
            sign_out <= 1'b0;
        end else begin
            done <= 1'b0;
            case (state)
                IDLE: begin
                    if (start) begin
                        if (b == '0) begin
                            result <= '0;
                            done   <= 1'b1;
                        end else begin
                            sign_out <= a[WORD-1] ^ b[WORD-1];
                            numer    <= {(a[WORD-1] ? -a : a), {FRAC_BITS{1'b0}}};
                            denom    <= {{FRAC_BITS{1'b0}}, (b[WORD-1] ? -b : b)};
                            quotient <= '0;
                            remainder<= '0;
                            count    <= WIDE[$clog2(WIDE):0];
                            state    <= BUSY;
                        end
                    end
                end
                BUSY: begin
                    if (rem_next >= denom) begin
                        remainder         <= rem_next - denom;
                        quotient[count-1] <= 1'b1;
                    end else begin
                        remainder         <= rem_next;
                        quotient[count-1] <= 1'b0;
                    end
                    count <= count - 1;
                    if (count == 1)
                        state <= FINISH;
                end
                FINISH: begin
                    result <= sign_out ? -quotient[WORD-1:0] : quotient[WORD-1:0];
                    done   <= 1'b1;
                    state  <= IDLE;
                end
                default: state <= IDLE;
            endcase
        end
    end
endmodule

// fp_inv: 1/b via fp_div with numerator = ONE. Stages C/F/H share one
// reciprocal across 3 numerators - collapses 3 fp_divs to 1 fp_inv + 3 muls.
module fp_inv #(
    parameter FRAC_BITS = 13,
    parameter WORD      = 27
)(
    input  logic                   clk,    // clock
    input  logic                   rst_n,  // active-low reset
    input  logic                   start,  // 1-cycle pulse: latch b, begin inversion
    input  logic signed [WORD-1:0] b,      // divisor (Q14.13); b==0 → result=0
    output logic signed [WORD-1:0] result, // 1/b (Q14.13), valid when done=1
    output logic                   done    // 1-cycle pulse 41c after start
);
    localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
    fp_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_div (
        .clk(clk), .rst_n(rst_n), .start(start),
        .a(ONE), .b(b), .result(result), .done(done)
    );
```

```
endmodule


module rtl_dma_master
    import raytracer_pkg::*;
#(
    parameter int N           = 480,
    parameter int WIDTH       = 480,
    parameter int FRAME_BYTES = WIDTH * 360 * 4
)(
    input  logic                     clk,                // clock
    input  logic                     reset,              // active-HIGH sync reset
    /* CSR slave bus (only DMA_CTRL writes are decoded here). */
    input  logic                     chipselect,         // slave chip-select
    input  logic                     write,              // slave write strobe
    input  logic [9:0]               address,            // slave word address
    input  logic [31:0]              writedata,          // slave write data (bit0=ENABLE, 1=CLEAR_ERR, 2=
CLEAR_DONE)
    /* From frame FSM */
    input  logic                     start_pulse,        // batch-start kick; clears dma_done_sticky
    input  logic                     core_done,          // compute pipeline done; triggers dma_kick when enabled
    input  logic [$clog2(WIDTH)-1:0] pixel_x_reg,        // pixel-x of current batch (for addr_burst computation)
    input  logic [8:0]               pixel_y_reg,        // pixel-y of current batch (row index)
    input  logic                     bank_sel,           // compute's BRAM bank - DMA reads the other (~bank_sel)
    input  logic                     write_slot,         // 0/1 = DDR3 frame ring slot for this frame
    /* CSR-staged frame-buffer base address (written via the slave). */
    input  logic [31:0]              frame_buf_base_reg, // base DDR3 phys addr of 2-slot frame ring
    /* BRAM read port (color_mem provides the data; we drive the addresses). */
    output logic [$clog2(N)-1:0]     dma_rd_addr_lo,     // BRAM read addr for even-lane pixel (low 32 bits of
beat)
    output logic [$clog2(N)-1:0]     dma_rd_addr_hi,     // BRAM read addr for odd-lane pixel  (high 32 bits of
beat)
    input  logic [23:0]              dma_rd_data_lo,     // even-lane pixel data (24-bit BGR from color_mem)
    input  logic [23:0]              dma_rd_data_hi,     // odd-lane pixel data
    output logic [31:0]              m_avm_address,      // Avalon-MM master: burst base address (byte)
    output logic [7:0]               m_avm_byteenable,   // Avalon-MM master: byte enables (hardcoded 0xFF)
    output logic                     m_avm_write,        // Avalon-MM master: held high during 240-beat burst
    output logic [63:0]              m_avm_writedata,    // Avalon-MM master: {pad,hi_pixel,pad,lo_pixel}
    output logic [7:0]               m_avm_burstcount,   // Avalon-MM master: hardcoded 240 beats per burst
    input  logic                     m_avm_waitrequest,  // Avalon-MM master: bridge back-pressure (hold beat)
    output logic                     dma_busy,           // 1 while a burst is in flight; goes to DMA_STATUS bit0
    output logic                     dma_err,            // sticky error flag; goes to DMA_STATUS bit1
    output logic                     dma_done_sticky,    // last burst done; cleared on next core_done or CSR
    output logic                     dma_bank_sel,       // BRAM read bank  ( ~bank_sel, latched at burst start)
    output logic [2:0]               dma_dbg_state,      // FSM state for DMA_STATUS [5:3]
    output logic [7:0]               dma_dbg_beats       // beats remaining for DMA_STATUS [13:6]
);
    typedef enum logic [2:0] {
        DMA_IDLE,
        DMA_PREFETCH,
        DMA_WRITE,
        DMA_COMMIT,
        DMA_DONE
    } dma_state_t;

    dma_state_t              dma_state;
    logic [31:0]             addr_burst;
    logic [$clog2(N+1)-1:0]  lane_idx;
    logic [7:0]              beats_remaining;
    logic [3:0]              commit_delay;
    logic                    dma_enable;
    logic                    dma_kick;

    assign dma_dbg_state = dma_state;
    assign dma_dbg_beats = beats_remaining;

    // BRAM read addresses: pre-fetch the NEXT pair when accepting a beat;
    // hold the CURRENT pair under waitrequest so writedata stays stable.
    always_comb begin
        if (dma_state == DMA_WRITE && !m_avm_waitrequest) begin
            dma_rd_addr_lo = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(2);
            dma_rd_addr_hi = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(3);
        end else begin
            dma_rd_addr_lo = lane_idx[$clog2(N)-1:0];
```

```
            dma_rd_addr_hi = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(1);
        end
    end

    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            dma_state       <= DMA_IDLE;
            addr_burst      <= '0;
            lane_idx        <= '0;
            beats_remaining <= '0;
            dma_busy        <= 1'b0;
            dma_err         <= 1'b0;
            dma_done_sticky <= 1'b0;
            dma_bank_sel    <= 1'b0;
            commit_delay    <= '0;
            dma_enable      <= 1'b0;
            dma_kick        <= 1'b0;
        end else begin
            dma_kick <= core_done & dma_enable;

            // dma_done_sticky clears on either a new kick or on core_done.
            // The core_done clear MUST lead dma_kick by one cycle -
            // clearing on dma_kick instead races FRAME_DRAIN.
            if (start_pulse) dma_done_sticky <= 1'b0;
            if (core_done)   dma_done_sticky <= 1'b0;

            if (chipselect && write && address == ADDR_DMA_CTRL) begin
                dma_enable <= writedata[0];
                if (writedata[1]) dma_err         <= 1'b0;
                if (writedata[2]) dma_done_sticky <= 1'b0;
            end

            case (dma_state)
                DMA_IDLE: if (dma_kick) begin
                    dma_bank_sel    <= bank_sel;
                    addr_burst      <= frame_buf_base_reg
                                       + (write_slot ? FRAME_BYTES[31:0] : 32'd0)
                                       + ((pixel_y_reg * WIDTH + pixel_x_reg) << 2);
                    lane_idx        <= '0;
                    beats_remaining <= 8'd240;
                    dma_busy        <= 1'b1;
                    dma_state       <= DMA_PREFETCH;
                end

                DMA_PREFETCH: dma_state <= DMA_WRITE;

                DMA_WRITE: if (!m_avm_waitrequest) begin
                    lane_idx        <= lane_idx + ($clog2(N+1))'(2);
                    beats_remaining <= beats_remaining - 8'd1;
                    if (beats_remaining == 8'd1) begin
                        commit_delay <= 4'd15;
                        dma_state    <= DMA_COMMIT;
                    end
                end

                DMA_COMMIT: begin
                    if (commit_delay == 4'd0) dma_state <= DMA_DONE;
                    else                      commit_delay <= commit_delay - 4'd1;
                end

                DMA_DONE: begin
                    dma_busy        <= 1'b0;
                    dma_done_sticky <= 1'b1;
                    dma_state       <= DMA_IDLE;
                end

                default: dma_state <= DMA_IDLE;
            endcase
        end
    end

    assign m_avm_address    = addr_burst;
    assign m_avm_byteenable = 8'hFF;
    assign m_avm_writedata  = {8'b0, dma_rd_data_hi, 8'b0, dma_rd_data_lo};
```

```
    assign m_avm_burstcount = 8'd240;
    assign m_avm_write      = (dma_state == DMA_WRITE);
endmodule

// ============================================================================
// sun_threshold_compute - produces K_const = |light|² · 63/64, the threshold
// used by rtl_hit_resolve's sun-billboard test ((d·light)²  K_const·|d|²).
// Time-muxes one fp_mul across a 3-phase rotation (lx², ly², lz²); the 63/64
// fold is x - (x>>>6) (no DSP). Output updates continuously as light_{x,y,z}
// change. Cold-reset value matches light = (0, 1, 0) so the sun is
// unreachable until SW writes a real light direction.
// ============================================================================
module sun_threshold_compute #(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27
)(
    input  logic                  clk,
    input  logic                  reset,
    input  logic signed [WORD-1:0] light_x,
    input  logic signed [WORD-1:0] light_y,
    input  logic signed [WORD-1:0] light_z,
    output logic signed [WORD-1:0] K_const
);
    localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;

    logic [1:0] kc_phase;
    always_ff @(posedge clk or posedge reset) begin
        if (reset)                  kc_phase <= 2'd0;
        else if (kc_phase == 2'd2)  kc_phase <= 2'd0;
        else                        kc_phase <= kc_phase + 2'd1;
    end

    logic signed [WORD-1:0] kc_mul_a, kc_mul_b;
    always_comb begin
        case (kc_phase)
            2'd0:    begin kc_mul_a = light_x; kc_mul_b = light_x; end
            2'd1:    begin kc_mul_a = light_y; kc_mul_b = light_y; end
            default: begin kc_mul_a = light_z; kc_mul_b = light_z; end
        endcase
    end

    logic signed [WORD-1:0] kc_mul_result;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_l_sq (
        .clk(clk), .a(kc_mul_a), .b(kc_mul_b), .result(kc_mul_result)
    );

    logic signed [WORD-1:0] lx_sq_r, ly_sq_r, lz_sq_r;
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            lx_sq_r <= '0;
            ly_sq_r <= ONE_FX;
            lz_sq_r <= '0;
        end else begin
            case (kc_phase)
                2'd1:    lx_sq_r <= kc_mul_result;
                2'd2:    ly_sq_r <= kc_mul_result;
                default: lz_sq_r <= kc_mul_result;
            endcase
        end
    end

    logic signed [WORD-1:0] light_sq_w;
    assign light_sq_w = lx_sq_r + ly_sq_r + lz_sq_r;

    // K_const = light_sq - (light_sq >>> 6)  light_sq · 63/64.
    always_ff @(posedge clk or posedge reset) begin
        if (reset) K_const <= ONE_FX - (ONE_FX >>> 6);
        else       K_const <= light_sq_w - (light_sq_w >>> 6);
    end
endmodule

module raytracer_batch_mm
    import raytracer_pkg::*;
#(
```

```
    parameter int N         = 480,
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int WIDTH     = 480,
    parameter int HEIGHT    = 360
)(
    input  logic                     clk,             // divclk (125 MHz target)
    input  logic                     reset,           // active-HIGH sync reset (inverted internally to rst_n)

    input  logic                     chipselect,      // slave selected
    input  logic                     read,            // master asserts to begin a read transaction
    input  logic                     write,           // master asserts to begin a write transaction
    input  logic [9:0]               address,         // word address (1024-word window)
    input  logic [31:0]              writedata,       // write data; HW masks each CSR's relevant bits
    output logic [31:0]              readdata,        // read data; fixed 2-cycle latency from read assertion

    // Avalon-MM master → hps_0.f2h_axi_slave (qsys-bridged to AXI3)
    output logic [31:0]              m_avm_address,    // burst base byte address (DDR3 phys addr)
    output logic [7:0]               m_avm_byteenable, // byte enables (hardcoded 0xFF)
    output logic                     m_avm_write,      // held high during DMA burst
    output logic [63:0]              m_avm_writedata,  // 64-bit beat = 2 packed BGRX pixels
    output logic [7:0]               m_avm_burstcount, // hardcoded 240 beats per burst
    input  logic                     m_avm_waitrequest // back-pressure from f2h bridge
);

    /*
     * Register map (word offsets - addresses live in raytracer_pkg as ADDR_*).
     * SW writes land directly on the live regs; the per-frame loop is
     * WAIT_FRAME → write all CSRs → FRAME_START so writes never tear an
     * in-flight frame. Most fields are signed Q FRAC_BITS.
     *
     *   0    CONTROL              bit2 frame_start, bit3 clear frame_done
     *   1    STATUS               bit2 frame_done, bit3 frame_running, bit4 write_slot
     *   4-6  LIGHT_{X,Y,Z}        light vector (also drives sun via |light|·63/64)
     *   7-8  R{0,1}_SQ            sphere radii² (SW pre-squares r)
     *   9    DMA_CTRL             bit0 enable, bit1 clear_err, bit2 clear_done
     *  10    DMA_STATUS           [0] busy, [1] err, [2] done, [5:3] state, [13:6] beats
     *  11    reserved (formerly LIGHT_COMMIT; not needed post-collapse)
     *  15    FRAME_BUF_BASE       DDR3 phys addr; HW writes pix_y·W+pix_x at slot offset
     *  16-18 SC1_{X,Y,Z}          sphere 1 (red) center
     *  19    MAX_DEPTH            3-bit recursion depth (default 3; 0 disables refl)
     *  20-21 REFLECT_SHIFT_{0,1}  5-bit per-sphere bounce attenuation (>>SHIFT)
     *  22-23 CAM_{X,Z}            camera origin (cam_y locked at 1.5)
     *  24-31 RIGHT/UP/FWD_*       precomputed basis from SW (right.y=0, no roll)
     *  32-34 SC0_{X,Y,Z}          sphere 0 (blue) center
     *  35-40 PC{1,2}_{R,G,B}      floor checker colors
     *  41-46 SCOL{0,1}_{R,G,B}    per-sphere base colors
     *  47    FLOOR_MODE           00=1m checker, 01=Sierpinski, 10=stripes, 11=0.5m
     *  48-49 FLOOR_LIM_{X,Z}      bounded plane half-extents (default max-pos)
     *  50-52 CONE_{X,Y,Z}         cone apex (default origin)
     *  53    CONE_K_SQ            tan²(half-angle) Q14.13 (default 1.0 = 45°)
     *  54    CONE_HEIGHT          finite cone height (default 0 = disabled)
     *  55-57 CONE_COL_{R,G,B}     cone surface color (default mid-grey)
     *  58    REFLECT_SHIFT_CONE   5-bit cone bounce attenuation (default 0 matte)
     *  59-63 reserved (former sun CSRs; sun is derived from light/|light| now)
     *  64..  COLOR_0..N           BRAM mirror, packed {00,R,G,B} (DMA is the
     *                             primary readback path)
     */

    logic [$clog2(WIDTH)-1:0]  pixel_x_reg;
    logic [$clog2(HEIGHT)-1:0] pixel_y_reg;
    logic signed [WORD-1:0]    light_x_reg;
    logic signed [WORD-1:0]    light_y_reg;
    logic signed [WORD-1:0]    light_z_reg;

    // Reflection / recursion control
    localparam int             MAX_DEPTH_W     = 3;
    localparam int             REFLECT_SHIFT_W = 5;
    localparam int             DEFAULT_MAX_DEPTH       = 3;
    localparam int             DEFAULT_REFLECT_SHIFT_0 = 2;  // 25%
    localparam int             DEFAULT_REFLECT_SHIFT_1 = 1;  // 50%
    logic [MAX_DEPTH_W-1:0]     max_depth_reg;

    // Camera state (cam_y locked in HW at CAM_Y_FIXED; right.y hardcoded 0).
```

```
    logic signed [WORD-1:0]    cam_x_reg,    cam_z_reg;
    logic signed [WORD-1:0]    right_x_reg,  right_z_reg;
    logic signed [WORD-1:0]    up_x_reg,     up_y_reg,     up_z_reg;
    logic signed [WORD-1:0]    fwd_x_reg,    fwd_y_reg,    fwd_z_reg;

    // Scene primitives - struct-typed CSR storage. Each field below maps
    // 1:1 to a CSR address; the case-arms inside the CSR write block
    // assign to sc0_reg.x, sc0_reg.color.r, sc0_reg.reflect_shift, etc.
    sphere_t sc0_reg, sc1_reg;
    cone_t   cone_reg;
    floor_t  floor_reg;

    logic                      start_pulse;

    // Frame FSM: →KICKWAIT(core_done)→KICK loop, then DRAIN(dma_done) →
    // DONE_PULSE
    typedef enum logic [2:0] {
        FRAME_IDLE,
        FRAME_KICK,
        FRAME_WAIT,
        FRAME_DRAIN,
        FRAME_DONE_PULSE
    } frame_state_t;

    frame_state_t              frame_state;
    logic [$clog2(WIDTH)-1:0]  pix_x_gen;
    logic [$clog2(HEIGHT+1)-1:0] pix_y_gen;     // 0..HEIGHT (post-last-advance)
    logic                      frame_running_reg;
    logic                      frame_done_sticky;
    logic [31:0]               frame_buf_base_reg;

    // Frame-level double buffer. Driver allocates 2·FRAME_BYTES at
    // FRAME_BUF_BASE; HW alternates slots so SW reads one while HW writes
    // the other. write_slot toggles only at FRAME_DONE_PULSE.
    localparam int FRAME_BYTES = WIDTH * HEIGHT * 4;
    logic                      write_slot;

    logic                      core_done;
    logic                      core_wr_en;
    logic [$clog2(N)-1:0]      core_wr_addr;
    logic [23:0]               core_wr_data;

    logic                      dma_busy;
    logic                      dma_err;
    logic                      dma_done_sticky;
    logic                      dma_bank_sel;
    logic [2:0]                dma_dbg_state;
    logic [7:0]                dma_dbg_beats;
    logic [$clog2(N)-1:0]      dma_rd_addr_lo, dma_rd_addr_hi;
    logic [23:0]               dma_rd_data_lo, dma_rd_data_hi;

    localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;
    localparam signed [WORD-1:0] DEFAULT_SC0_X = 0;
    localparam signed [WORD-1:0] DEFAULT_SC0_Y = ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_SC0_Z = 0;
    localparam signed [WORD-1:0] DEFAULT_R0_SQ = ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_SC1_X = (5 * ONE_FX) / 2;
    localparam signed [WORD-1:0] DEFAULT_SC1_Y = ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_SC1_Z = 0;
    localparam signed [WORD-1:0] DEFAULT_R1_SQ = ONE_FX;

    localparam signed [WORD-1:0] CAM_Y_FIXED       = (3 * ONE_FX) / 2;
    localparam signed [WORD-1:0] DEFAULT_CAM_X     = '0;
    localparam signed [WORD-1:0] DEFAULT_CAM_Z     = -8 * ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_RIGHT_X   = -ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_RIGHT_Z   = '0;
    localparam signed [WORD-1:0] DEFAULT_UP_X      = '0;
    localparam signed [WORD-1:0] DEFAULT_UP_Y      = ONE_FX;
    localparam signed [WORD-1:0] DEFAULT_UP_Z      = '0;
    localparam signed [WORD-1:0] DEFAULT_FWD_X     = '0;
    localparam signed [WORD-1:0] DEFAULT_FWD_Y     = '0;
    localparam signed [WORD-1:0] DEFAULT_FWD_Z     = ONE_FX;

    localparam signed [WORD-1:0] DEFAULT_PC1       = (9 * ONE_FX) / 10;
```

```
    localparam signed [WORD-1:0] DEFAULT_PC2       = (4 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL0_R   = (2 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL0_G   = (3 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL0_B   = (8 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL1_R   = (8 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL1_G   = (3 * ONE_FX) / 10;
    localparam signed [WORD-1:0] DEFAULT_SCOL1_B   = (2 * ONE_FX) / 10;
    localparam logic [1:0]       DEFAULT_FLOOR_MODE = 2'b00;
    localparam signed [WORD-1:0] DEFAULT_FLOOR_LIM = {1'b0, {(WORD-1){1'b1}}};

    // Sun-billboard threshold K_const = |light|² · 63/64. Time-muxed
    // single-DSP compute lives in sun_threshold_compute.
    logic signed [WORD-1:0] K_const_r;
    sun_threshold_compute #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sun_threshold (
        .clk     (clk),
        .reset   (reset),
        .light_x (light_x_reg),
        .light_y (light_y_reg),
        .light_z (light_z_reg),
        .K_const (K_const_r)
    );

    // Bundle the four primitive registers into a single scene_t for the
    // pipeline. Pure struct-literal - Quartus prunes unused fields at each
    // consumer stage.
    scene_t scene_active;
    assign scene_active = '{s0:    sc0_reg,
                            s1:    sc1_reg,
                            cone:  cone_reg,
                            floor: floor_reg};

    rtl_batch_pipe #(
        .K         (4),
        .BATCH     (N),
        .FRAC_BITS (FRAC_BITS),
        .WORD      (WORD),
        .WIDTH     (WIDTH),
        .HEIGHT    (HEIGHT)
    ) core_inst (
        .clk        (clk),
        .rst_n      (~reset),
        .start      (start_pulse),
        .pixel_x    (pixel_x_reg),
        .pixel_y    (pixel_y_reg),
        .light_x    (light_x_reg),
        .light_y    (light_y_reg),
        .light_z    (light_z_reg),
        .scene      (scene_active),
        .max_depth  (max_depth_reg),
        .cam_x      (cam_x_reg),
        .cam_y      (CAM_Y_FIXED),
        .cam_z      (cam_z_reg),
        .right_x    (right_x_reg),
        .right_z    (right_z_reg),
        .up_x       (up_x_reg),
        .up_y       (up_y_reg),
        .up_z       (up_z_reg),
        .fwd_x      (fwd_x_reg),
        .fwd_y      (fwd_y_reg),
        .fwd_z      (fwd_z_reg),
        .K_const    (K_const_r),
        .done       (core_done),
        .wr_en      (core_wr_en),
        .wr_addr    (core_wr_addr),
        .wr_data    (core_wr_data)
    );

    logic        bank_sel;
    localparam int COLOR_MEM_DEPTH = 1 << ($clog2(N) + 1);
    (* ramstyle = "no_rw_check, M10K" *)
    logic [23:0] color_mem [0:COLOR_MEM_DEPTH-1];

    always_ff @(posedge clk or posedge reset) begin
        if (reset) bank_sel <= 1'b0;
```

```
        else if (start_pulse) bank_sel <= ~bank_sel;
    end

    always_ff @(posedge clk) begin
        if (core_wr_en)
            color_mem[{bank_sel, core_wr_addr}] <= core_wr_data;
    end

    // ============================================================================
    // CSR data-register bank - direct-mapped Avalon-MM slave writes.
    // Each register is one FF; writes land at the matching address. Writes
    // to ADDR_CONTROL (frame_start / clear_done) and ADDR_DMA_CTRL (DMA
    // enable) are NOT data writes - they live in the Frame FSM block
    // below and in rtl_dma_master respectively.
    // ============================================================================
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            light_x_reg     <= '0;
            // light_y resets to ONE_FX so |light|² = 1 and K_const  63/64.
            // Without this, K_const = 0 and the sun overruns the sky until
            // SW writes a real light direction.
            light_y_reg     <= ONE_FX;
            light_z_reg     <= '0;
            max_depth_reg   <= DEFAULT_MAX_DEPTH[MAX_DEPTH_W-1:0];
            cam_x_reg       <= DEFAULT_CAM_X;
            cam_z_reg       <= DEFAULT_CAM_Z;
            right_x_reg     <= DEFAULT_RIGHT_X;
            right_z_reg     <= DEFAULT_RIGHT_Z;
            up_x_reg        <= DEFAULT_UP_X;
            up_y_reg        <= DEFAULT_UP_Y;
            up_z_reg        <= DEFAULT_UP_Z;
            fwd_x_reg       <= DEFAULT_FWD_X;
            fwd_y_reg       <= DEFAULT_FWD_Y;
            fwd_z_reg       <= DEFAULT_FWD_Z;
            // Sphere 0 (blue, 25% reflective by default)
            sc0_reg <= '{x:    DEFAULT_SC0_X,
                         y:    DEFAULT_SC0_Y,
                         z:    DEFAULT_SC0_Z,
                         r_sq: DEFAULT_R0_SQ,
                         color: '{r: DEFAULT_SCOL0_R, g: DEFAULT_SCOL0_G, b: DEFAULT_SCOL0_B},
                         reflect_shift: DEFAULT_REFLECT_SHIFT_0[REFLECT_SHIFT_W-1:0]};
            // Sphere 1 (red, 50% reflective by default)
            sc1_reg <= '{x:    DEFAULT_SC1_X,
                         y:    DEFAULT_SC1_Y,
                         z:    DEFAULT_SC1_Z,
                         r_sq: DEFAULT_R1_SQ,
                         color: '{r: DEFAULT_SCOL1_R, g: DEFAULT_SCOL1_G, b: DEFAULT_SCOL1_B},
                         reflect_shift: DEFAULT_REFLECT_SHIFT_1[REFLECT_SHIFT_W-1:0]};
            // Cone - height=0 disables (slab empty); SW writes a real height
            // to enable. Default mid-grey, matte, k²=1.0 (45° half-angle).
            cone_reg <= '{apex_x: '0, apex_y: '0, apex_z: '0,
                          k_sq:   27'sd8192,                // Q14.13: 1.0
                          height: '0,
                          norm_factor: 27'sd11585,          // sqrt(2)
                          color: '{r: 27'sd4096, g: 27'sd4096, b: 27'sd4096},  // 0.5,0.5,0.5
                          reflect_shift: '0};
            // Floor (checker, max half-extents by default)
            floor_reg <= '{pc1: '{r: DEFAULT_PC1, g: DEFAULT_PC1, b: DEFAULT_PC1},
                           pc2: '{r: DEFAULT_PC2, g: DEFAULT_PC2, b: DEFAULT_PC2},
                           mode: DEFAULT_FLOOR_MODE,
                           lim_x: DEFAULT_FLOOR_LIM,
                           lim_z: DEFAULT_FLOOR_LIM};
            frame_buf_base_reg <= '0;
        end else if (chipselect && write) begin
            case (address)
                ADDR_LIGHT_X:   light_x_reg <= writedata[WORD-1:0];
                ADDR_LIGHT_Y:   light_y_reg <= writedata[WORD-1:0];
                ADDR_LIGHT_Z:   light_z_reg <= writedata[WORD-1:0];
                ADDR_FRAME_BUF_BASE: frame_buf_base_reg <= writedata;
                ADDR_MAX_DEPTH:      max_depth_reg <= writedata[MAX_DEPTH_W-1:0];
                ADDR_CAM_X:          cam_x_reg     <= writedata[WORD-1:0];
                ADDR_CAM_Z:          cam_z_reg     <= writedata[WORD-1:0];
                ADDR_RIGHT_X:        right_x_reg   <= writedata[WORD-1:0];
                ADDR_RIGHT_Z:        right_z_reg   <= writedata[WORD-1:0];
```

```
            ADDR_UP_X:           up_x_reg      <= writedata[WORD-1:0];
            ADDR_UP_Y:           up_y_reg      <= writedata[WORD-1:0];
            ADDR_UP_Z:           up_z_reg      <= writedata[WORD-1:0];
            ADDR_FWD_X:          fwd_x_reg     <= writedata[WORD-1:0];
            ADDR_FWD_Y:          fwd_y_reg     <= writedata[WORD-1:0];
            ADDR_FWD_Z:          fwd_z_reg     <= writedata[WORD-1:0];
            // Sphere 0
            ADDR_SC0_X:              sc0_reg.x             <= writedata[WORD-1:0];
            ADDR_SC0_Y:              sc0_reg.y             <= writedata[WORD-1:0];
            ADDR_SC0_Z:              sc0_reg.z             <= writedata[WORD-1:0];
            ADDR_R0_SQ:              sc0_reg.r_sq          <= writedata[WORD-1:0];
            ADDR_SCOL0_R:            sc0_reg.color.r       <= writedata[WORD-1:0];
            ADDR_SCOL0_G:            sc0_reg.color.g       <= writedata[WORD-1:0];
            ADDR_SCOL0_B:            sc0_reg.color.b       <= writedata[WORD-1:0];
            ADDR_REFLECT_SHIFT_0:    sc0_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
            // Sphere 1
            ADDR_SC1_X:              sc1_reg.x             <= writedata[WORD-1:0];
            ADDR_SC1_Y:              sc1_reg.y             <= writedata[WORD-1:0];
            ADDR_SC1_Z:              sc1_reg.z             <= writedata[WORD-1:0];
            ADDR_R1_SQ:              sc1_reg.r_sq          <= writedata[WORD-1:0];
            ADDR_SCOL1_R:            sc1_reg.color.r       <= writedata[WORD-1:0];
            ADDR_SCOL1_G:            sc1_reg.color.g       <= writedata[WORD-1:0];
            ADDR_SCOL1_B:            sc1_reg.color.b       <= writedata[WORD-1:0];
            ADDR_REFLECT_SHIFT_1:    sc1_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
            // Cone
            ADDR_CONE_X:             cone_reg.apex_x       <= writedata[WORD-1:0];
            ADDR_CONE_Y:             cone_reg.apex_y       <= writedata[WORD-1:0];
            ADDR_CONE_Z:             cone_reg.apex_z       <= writedata[WORD-1:0];
            ADDR_CONE_K_SQ:          cone_reg.k_sq         <= writedata[WORD-1:0];
            ADDR_CONE_HEIGHT:        cone_reg.height       <= writedata[WORD-1:0];
            ADDR_CONE_NORM_FACTOR:   cone_reg.norm_factor  <= writedata[WORD-1:0];
            ADDR_CONE_COL_R:         cone_reg.color.r      <= writedata[WORD-1:0];
            ADDR_CONE_COL_G:         cone_reg.color.g      <= writedata[WORD-1:0];
            ADDR_CONE_COL_B:         cone_reg.color.b      <= writedata[WORD-1:0];
            ADDR_REFLECT_SHIFT_CONE: cone_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
            // Floor
            ADDR_PC1_R:              floor_reg.pc1.r       <= writedata[WORD-1:0];
            ADDR_PC1_G:              floor_reg.pc1.g       <= writedata[WORD-1:0];
            ADDR_PC1_B:              floor_reg.pc1.b       <= writedata[WORD-1:0];
            ADDR_PC2_R:              floor_reg.pc2.r       <= writedata[WORD-1:0];
            ADDR_PC2_G:              floor_reg.pc2.g       <= writedata[WORD-1:0];
            ADDR_PC2_B:              floor_reg.pc2.b       <= writedata[WORD-1:0];
            ADDR_FLOOR_MODE:         floor_reg.mode        <= writedata[1:0];
            ADDR_FLOOR_LIM_X:        floor_reg.lim_x       <= writedata[WORD-1:0];
            ADDR_FLOOR_LIM_Z:        floor_reg.lim_z       <= writedata[WORD-1:0];
            default: begin end
        endcase
    end
end

// ============================================================================
// Frame FSM - chains per-batch →→→KICKWAITDRAINDONE_PULSE for an entire
// frame on a single SW kick. ADDR_CONTROL writes (frame_start /
// clear_done) are handled here; they mutate FSM state, not data regs.
// write_slot flips at FRAME_DONE_PULSE for frame-level double buffering.
// ============================================================================
always_ff @(posedge clk or posedge reset) begin
    if (reset) begin
        pixel_x_reg       <= '0;
        pixel_y_reg       <= '0;
        start_pulse       <= 1'b0;
        frame_state       <= FRAME_IDLE;
        pix_x_gen         <= '0;
        pix_y_gen         <= '0;
        frame_running_reg <= 1'b0;
        frame_done_sticky <= 1'b0;
        write_slot        <= 1'b0;
    end else begin
        start_pulse <= 1'b0;

        // ADDR_CONTROL writes: bit3 clears frame_done_sticky, bit2 kicks
        // a new frame. Handled here (not in the CSR block) because every
        // signal touched is FSM state.
        if (chipselect && write && address == ADDR_CONTROL) begin
```

```
            if (writedata[3]) frame_done_sticky <= 1'b0;
            if (writedata[2]) begin
                pix_x_gen         <= '0;
                pix_y_gen         <= '0;
                frame_running_reg <= 1'b1;
                frame_done_sticky <= 1'b0;
                frame_state       <= FRAME_KICK;
            end
        end

        case (frame_state)
            FRAME_IDLE: begin
                /* wait for SW to write CONTROL[2] (handled above) */
            end
            FRAME_KICK: begin
                // Latch this batch's pix coords (pre-edge - pix_gen
                // holds the NEXT batch); fire start_pulse.
                pixel_x_reg <= pix_x_gen;
                pixel_y_reg <= pix_y_gen[$clog2(HEIGHT)-1:0];
                start_pulse <= 1'b1;

                if (pix_x_gen + N >= WIDTH) begin
                    pix_x_gen <= '0;
                    pix_y_gen <= pix_y_gen + 1;
                end else begin
                    pix_x_gen <= pix_x_gen + N;
                end

                frame_state <= FRAME_WAIT;
            end
            // Advance on core_done (NOT dma_done_sticky) so batch N's
            // DMA overlaps with batch N+1's compute.
            FRAME_WAIT: if (core_done) begin
                if (pix_y_gen >= HEIGHT)
                    frame_state <= FRAME_DRAIN;
                else
                    frame_state <= FRAME_KICK;
            end
            FRAME_DRAIN: if (dma_done_sticky)
                frame_state <= FRAME_DONE_PULSE;
            FRAME_DONE_PULSE: begin
                frame_running_reg <= 1'b0;
                frame_done_sticky <= 1'b1;
                // Flip write_slot AFTER the frame's DMA committed so the
                // next frame writes the slot SW just released.
                write_slot        <= ~write_slot;
                frame_state       <= FRAME_IDLE;
            end
            default: frame_state <= FRAME_IDLE;
        endcase
    end
end

// Slave read path - single-beat, fixed readLatency=1. address_r
// absorbs the qsys-side IC route; the 3-way case mux drives readdata
// combinationally within the same period.
logic [9:0]  address_r;
always_ff @(posedge clk or posedge reset) begin
    if (reset) address_r <= '0;
    else       address_r <= address;
end

logic [31:0] ctrl_rdata;
always_comb begin
    case (address_r)
        ADDR_STATUS:         ctrl_rdata = {27'b0, write_slot,
                                           frame_running_reg,
                                           frame_done_sticky,
                                           2'b00};
        ADDR_DMA_STATUS:     ctrl_rdata = {18'b0, dma_dbg_beats,
                                           dma_dbg_state, dma_done_sticky,
                                           dma_err, dma_busy};
        ADDR_FRAME_BUF_BASE: ctrl_rdata = frame_buf_base_reg;
        default:             ctrl_rdata = 32'b0;
```

```
        endcase
    end

    assign readdata = ctrl_rdata;

    always_ff @(posedge clk) begin
        dma_rd_data_lo <= color_mem[{dma_bank_sel, dma_rd_addr_lo}];
    end
    always_ff @(posedge clk) begin
        dma_rd_data_hi <= color_mem[{dma_bank_sel, dma_rd_addr_hi}];
    end

    rtl_dma_master #(
        .N           (N),
        .WIDTH       (WIDTH),
        .FRAME_BYTES (FRAME_BYTES)
    ) u_dma (
        .clk               (clk),
        .reset             (reset),
        .chipselect        (chipselect),
        .write             (write),
        .address           (address),
        .writedata         (writedata),
        .start_pulse       (start_pulse),
        .core_done         (core_done),
        .pixel_x_reg       (pixel_x_reg),
        .pixel_y_reg       (pixel_y_reg),
        .bank_sel          (bank_sel),
        .write_slot        (write_slot),
        .frame_buf_base_reg (frame_buf_base_reg),
        .dma_rd_addr_lo    (dma_rd_addr_lo),
        .dma_rd_addr_hi    (dma_rd_addr_hi),
        .dma_rd_data_lo    (dma_rd_data_lo),
        .dma_rd_data_hi    (dma_rd_data_hi),
        .m_avm_address     (m_avm_address),
        .m_avm_byteenable  (m_avm_byteenable),
        .m_avm_write       (m_avm_write),
        .m_avm_writedata   (m_avm_writedata),
        .m_avm_burstcount  (m_avm_burstcount),
        .m_avm_waitrequest (m_avm_waitrequest),
        .dma_busy          (dma_busy),
        .dma_err           (dma_err),
        .dma_done_sticky   (dma_done_sticky),
        .dma_bank_sel      (dma_bank_sel),
        .dma_dbg_state     (dma_dbg_state),
        .dma_dbg_beats     (dma_dbg_beats)
    );

endmodule

// Stage A - RAY GEN: select raw direction (basis_compute output for primaries,
// latched parent reflection for spawns) and compute |raw|² for stage C's
// quadratic `a` coefficient. Also precomputes a_cone = |d|² - (1+k²)·D.y² so
// that u_inv_a_cone can kick at stage C's t=0 alongside u_inv_a/u_inv_pd -
// see docs/cone_fp_inv_timing.html.
module rtl_ray_gen
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int WIDTH     = 480,
    parameter int HEIGHT    = 360,
    parameter int TAG_W     = 7
)(
    input  logic                        clk, rst_n,               // clock, async neg-edge reset
    input  logic [5:0]                  tick,                      // pipe-local tick counter (0..II-1)
    input  pipe_baggage_t               in_bag,                    // {sv, tag, meta} - latched at tick=0
    // Reflection-ray override path: in_use_override=1 selects dir_*_lat
    input  logic                        in_use_override,           // 1=spawn (use in_dir), 0=primary (use basis
output)
    input  logic signed [WORD-1:0]     in_orig_x, in_orig_y, in_orig_z, // ray origin: camera for primaries, parent
 hit for spawns
    input  logic signed [WORD-1:0]     in_dir_x, in_dir_y, in_dir_z,    // override direction (parent reflection);
used iff in_use_override=1
```

```
    // Shared basis_compute output from rtl_batch_pipe - settles by issue+5.
    input  logic signed [WORD-1:0]     in_basis_raw_x, in_basis_raw_y, in_basis_raw_z, // primary raw direction (right·px + up·py + fwd)
    input  scene_t                     in_scene,                   // scene CSR relay; only in_scene.cone.k_sq is read here
    output logic signed [WORD-1:0]     raygen_raw_x, raygen_raw_y, raygen_raw_z, // captured raw direction → Stage C
    output logic signed [WORD-1:0]     raygen_mag_sq,              // |d|², used by sphere quadratic + sun billboard
    output logic signed [WORD-1:0]     raygen_a_cone,              // |d|² - (1+k²)·D.y² - feeds u_inv_a_cone at C-t=0
    output logic signed [WORD-1:0]     raygen_ox, raygen_oy, raygen_oz,  // registered origin forwarded to Stage C
    output pipe_baggage_t              raygen_bag                  // baggage forwarded to Stage C
);
    localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;

    pipe_baggage_t              bag_r;
    logic                       use_override_r;
    logic signed [WORD-1:0]     orig_x_r, orig_y_r, orig_z_r;
    logic signed [WORD-1:0]     dir_x_lat, dir_y_lat, dir_z_lat;
    logic signed [WORD-1:0]     raw_x_r, raw_y_r, raw_z_r;
    logic signed [WORD-1:0]     prod_x_r, prod_y_r, prod_z_r;
    logic signed [WORD-1:0]     mag_sq_r;
    logic signed [WORD-1:0]     one_plus_k_sq_r;
    logic signed [WORD-1:0]     kp1_dy_sq_r, a_cone_r;

    logic signed [WORD-1:0] mul_a, mul_b, mul_result;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));

    always_comb begin
        mul_a = '0; mul_b = '0;
        case (tick)
            6'd8:  begin mul_a = raw_x_r;    mul_b = raw_x_r;          end
            6'd9:  begin mul_a = raw_y_r;    mul_b = raw_y_r;          end
            6'd10: begin mul_a = raw_z_r;    mul_b = raw_z_r;          end
            6'd11: begin mul_a = prod_y_r;   mul_b = one_plus_k_sq_r;  end // kp1_dy_sq → cap t=12
            default: begin end
        endcase
    end

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            bag_r <= '0;
            use_override_r <= 1'b0;
            orig_x_r <= '0; orig_y_r <= '0; orig_z_r <= '0;
            dir_x_lat <= '0; dir_y_lat <= '0; dir_z_lat <= '0;
            raw_x_r <= '0; raw_y_r <= '0; raw_z_r <= '0;
            prod_x_r <= '0; prod_y_r <= '0; prod_z_r <= '0;
            mag_sq_r <= '0;
            one_plus_k_sq_r <= '0;
            kp1_dy_sq_r <= '0;
            a_cone_r <= '0;
        end else begin
            one_plus_k_sq_r <= in_scene.cone.k_sq + ONE_FX;

            case (tick)
                6'd0: begin
                    bag_r <= in_bag;
                    use_override_r <= in_use_override;
                    orig_x_r <= in_orig_x;
                    orig_y_r <= in_orig_y;
                    orig_z_r <= in_orig_z;
                    dir_x_lat <= in_dir_x;
                    dir_y_lat <= in_dir_y;
                    dir_z_lat <= in_dir_z;
                end
                6'd5: begin
                    raw_x_r <= use_override_r ? dir_x_lat : in_basis_raw_x;
                    raw_y_r <= use_override_r ? dir_y_lat : in_basis_raw_y;
                    raw_z_r <= use_override_r ? dir_z_lat : in_basis_raw_z;
                end
                6'd9:  prod_x_r <= mul_result;
                6'd10: prod_y_r <= mul_result;
                6'd11: prod_z_r <= mul_result;
```

```
                6'd12: begin
                    mag_sq_r    <= prod_x_r + prod_y_r + prod_z_r;
                    kp1_dy_sq_r <= mul_result;   // (1+k²)·D.y² from t=11 issue
                end
                6'd13: a_cone_r <= mag_sq_r - kp1_dy_sq_r;
                default: begin end
            endcase
        end
    end

    assign raygen_raw_x = raw_x_r;
    assign raygen_raw_y = raw_y_r;
    assign raygen_raw_z = raw_z_r;
    assign raygen_mag_sq = mag_sq_r;
    assign raygen_a_cone = a_cone_r;
    assign raygen_ox = orig_x_r;
    assign raygen_oy = orig_y_r;
    assign raygen_oz = orig_z_r;
    assign raygen_bag = bag_r;
endmodule

// Stage B - ISECT PREP: half-discriminant form for both spheres + plane denom.
//   a        = |raw|² (shared from stage A)
//   half_b   = dot(dir, oc),  oc = orig - sphere_center
//   c        = dot(oc, oc) - r²
//   disc_sqrt = sqrt(half_b² - a·c) (per sphere)
//   inv_a, inv_pd: emitted as reciprocals so stage C multiplies instead of
//                  divides (LAB-relax saves divs).
module rtl_isect_prep
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                      // clock, async neg-edge reset
    input  logic [5:0]              tick,                            // pipe-local tick counter (0..II-1)
    input  logic signed [WORD-1:0]  raygen_raw_x, raygen_raw_y, raygen_raw_z, // ray direction from Stage A
    input  logic signed [WORD-1:0]  raygen_mag_sq,                   // |d|² from Stage A (= sphere "a")
    input  logic signed [WORD-1:0]  raygen_a_cone,                   // a_cone = |d|⁻²(1+k²)·D.y² (precomputed in
 Stage A)
    // Ray origin (CAM for primaries, parent hit for reflection spawns).
    input  logic signed [WORD-1:0]  raygen_ox, raygen_oy, raygen_oz,
    input  scene_t                  in_scene,                        // scene relay (reads .s0/.s1/.cone geometry
)
    input  pipe_baggage_t           raygen_bag,                      // {sv, tag, meta} forwarded
    output logic signed [WORD-1:0]  isect_dx, isect_dy, isect_dz,    // registered direction → Stage C
    output logic signed [WORD-1:0]  isect_inv_sa,                    // 1/|d|² (1/a) - sphere t-solve via
multiply
    output logic signed [WORD-1:0]  isect_s0_nb,                     // sphere 0: -half_b (numerator term)
    output logic signed [WORD-1:0]  isect_s0_dsq,                    // sphere 0: √(half_b² - a·c)
    output logic                    isect_s0_nh,                     // sphere 0: no_hit flag (disc < 0)
    output logic signed [WORD-1:0]  isect_s1_nb,                     // sphere 1: -half_b
    output logic signed [WORD-1:0]  isect_s1_dsq,                    // sphere 1: √disc
    output logic                    isect_s1_nh,                     // sphere 1: no_hit flag
    output logic signed [WORD-1:0]  isect_inv_pd,                    // 1/dir.y for plane t-solve
    // Cone outputs. u_inv_a_cone kicks at t=0 (with a_cone_r from stage A);
    // result wire valid by t=42, captured at stage DE's t=0 boundary latch.
    output logic signed [WORD-1:0]  isect_cone_nb,                   // cone: -b_cone
    output logic signed [WORD-1:0]  isect_cone_dsq,                  // cone: √(b² - a·c)
    output logic                    isect_cone_nh,                   // cone: no_hit flag
    output logic signed [WORD-1:0]  isect_inv_a_cone,                // 1/a_cone - cone t-solve via multiply
    // Δ = origin - apex, forwarded for stage C's cone hit_y bounds check
    // and cap/normal math.
    output logic signed [WORD-1:0]  isect_cone_dx,                   // cone Δ.x = orig.x - apex.x
    output logic signed [WORD-1:0]  isect_cone_dy,                   // cone Δ.y (used for slab + cap)
    output logic signed [WORD-1:0]  isect_cone_dz,                   // cone Δ.z
    // Registered |d|² for stage C's sun squared-compare. Must read the
    // captured a_r (not raw raygen_mag_sq) since A is K=4 pixels ahead of B.
    output logic signed [WORD-1:0]  isect_sa,                        // registered |d|² for sun test
    // orig flows to C for hit-point reconstruction; orig_y for the plane numer.
    output logic signed [WORD-1:0]  isect_ox, isect_oy, isect_oz,    // registered origin → Stage C
    output pipe_baggage_t           isect_bag                        // baggage forwarded to Stage C
);
```

```
    localparam signed [WORD-1:0] ONE   = 1 <<< FRAC_BITS;

    logic signed [WORD-1:0] dx_r, dy_r, dz_r;
    logic signed [WORD-1:0] ox_r, oy_r, oz_r;
    logic signed [WORD-1:0] a_r;
    pipe_baggage_t          bag_r;

    sphere_t sc0_local, sc1_local;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            sc0_local <= '0;
            sc1_local <= '0;
        end else begin
            sc0_local <= in_scene.s0;
            sc1_local <= in_scene.s1;
        end
    end

    // oc = orig - sphere_center, one combinational subtract then registered.
    logic signed [WORD-1:0] oc0_x_c, oc0_y_c, oc0_z_c;
    logic signed [WORD-1:0] oc1_x_c, oc1_y_c, oc1_z_c;
    assign oc0_x_c = ox_r - sc0_local.x;
    assign oc0_y_c = oy_r - sc0_local.y;
    assign oc0_z_c = oz_r - sc0_local.z;
    assign oc1_x_c = ox_r - sc1_local.x;
    assign oc1_y_c = oy_r - sc1_local.y;
    assign oc1_z_c = oz_r - sc1_local.z;
    logic signed [WORD-1:0] oc0_x_r, oc0_y_r, oc0_z_r;
    logic signed [WORD-1:0] oc1_x_r, oc1_y_r, oc1_z_r;

    // Per-sphere products + intermediates
    logic signed [WORD-1:0] p0_hb0_r, p1_hb0_r, p2_hb0_r;
    logic signed [WORD-1:0] p0_oc0_r, p1_oc0_r, p2_oc0_r;
    logic signed [WORD-1:0] half_b0_r, c0_r;
    logic signed [WORD-1:0] half_b0_sq_r, a_c0_r;
    logic signed [WORD-1:0] disc_sqrt0_r;
    logic                   no_hit0_r;

    logic signed [WORD-1:0] p0_hb1_r, p1_hb1_r, p2_hb1_r;
    logic signed [WORD-1:0] p0_oc1_r, p1_oc1_r, p2_oc1_r;
    logic signed [WORD-1:0] half_b1_r, c1_r;
    logic signed [WORD-1:0] half_b1_sq_r, a_c1_r;
    logic signed [WORD-1:0] disc_sqrt1_r;
    logic                   no_hit1_r;

    // Cone state
    logic signed [WORD-1:0] cone_dx_r, cone_dy_r, cone_dz_r;       // Δ
    logic signed [WORD-1:0] cdx_sq_r, cdy_sq_r, cdz_sq_r;          // Δ.x², Δ.y², Δ.z²
    logic signed [WORD-1:0] k_cdy_sq_r;                            // Δk²·.y²
    logic signed [WORD-1:0] cdx_dx_r, cdy_dy_r, cdz_dz_r;          // Δ.{x,y,z}·D.{x,y,z}
    logic signed [WORD-1:0] k_cdy_dy_r;                            // k²·Δ(.y·D.y)
    logic signed [WORD-1:0] a_cone_r, b_cone_r, c_cone_r;
    logic signed [WORD-1:0] b_cone_sq_r, a_c_cone_r;
    logic signed [WORD-1:0] disc_sqrt_cone_r;
    logic                   no_hit_cone_r;

    // Two fp_muls: u_mul0 for sphere 0, u_mul1 for sphere 1.
    logic signed [WORD-1:0] m0_a, m0_b, m0_r;
    logic signed [WORD-1:0] m1_a, m1_b, m1_r;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul0 (.clk(clk), .a(m0_a), .b(m0_b), .result(m0_r));
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul1 (.clk(clk), .a(m1_a), .b(m1_b), .result(m1_r));

    logic signed [WORD-1:0] mc_a, mc_b, mc_r;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone (.clk(clk), .a(mc_a), .b(mc_b), .result(mc_r));

    // Locally-registered `tick` for the operand mux. Same pattern as
    // tick_local_h in rtl_hit_resolve: `tick` has 400+ destinations and
    // the cross-die IC to u_mul0/u_mul1/u_mul_cone selector clusters was
    // a top-violator class. This single FF drives the combined case
    // (no preserve so Quartus is free to place / replicate near the DSPs);
    // it lags `tick` by 1 cycle so case arm indices shift -1.
    logic [5:0] tick_local_i;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) tick_local_i <= 6'd0;
```

```
        else        tick_local_i <= tick;
    end

    always_comb begin
        m0_a = '0; m0_b = '0;
        m1_a = '0; m1_b = '0;
        mc_a = '0; mc_b = '0;
        case (tick_local_i)
            6'd1: begin                                                          // tick=2
                m0_a = dx_r;    m0_b = oc0_x_r;
                m1_a = dx_r;    m1_b = oc1_x_r;
                // mc free - D.y² and k²·D.y² now precomputed in stage A.
            end
            6'd2: begin                                                          // tick=3
                m0_a = dy_r;    m0_b = oc0_y_r;
                m1_a = dy_r;    m1_b = oc1_y_r;
            end
            6'd3: begin                                                          // tick=4
                m0_a = dz_r;    m0_b = oc0_z_r;
                m1_a = dz_r;    m1_b = oc1_z_r;
                mc_a = cone_dx_r; mc_b = cone_dx_r;    // Δ.x² → cap t=5
            end
            6'd4: begin                                                          // tick=5
                m0_a = oc0_x_r; m0_b = oc0_x_r;
                m1_a = oc1_x_r; m1_b = oc1_x_r;
                mc_a = cone_dy_r; mc_b = cone_dy_r;    // Δ.y² → cap t=6
            end
            6'd5: begin                                                          // tick=6
                m0_a = oc0_y_r; m0_b = oc0_y_r;
                m1_a = oc1_y_r; m1_b = oc1_y_r;
                mc_a = cone_dz_r; mc_b = cone_dz_r;    // Δ.z² → cap t=7
            end
            6'd6: begin                                                          // tick=7
                m0_a = oc0_z_r; m0_b = oc0_z_r;
                m1_a = oc1_z_r; m1_b = oc1_z_r;
                mc_a = in_scene.cone.k_sq; mc_b = cdy_sq_r;  // Δk²·.y² → cap t=8
            end
            6'd7: begin                                                          // tick=8
                m0_a = half_b0_r; m0_b = half_b0_r;
                m1_a = half_b1_r; m1_b = half_b1_r;
                mc_a = cone_dx_r; mc_b = dx_r;          // Δ.x·D.x → cap t=9
            end
            6'd8: begin                                                          // tick=9
                mc_a = cone_dy_r; mc_b = dy_r;          // Δ.y·D.y → cap t=10
            end
            6'd9: begin                                                          // tick=10
                m0_a = a_r; m0_b = c0_r;
                m1_a = a_r; m1_b = c1_r;
                mc_a = cone_dz_r; mc_b = dz_r;          // Δ.z·D.z → cap t=11
            end
            6'd10: begin                                                         // tick=11
                mc_a = in_scene.cone.k_sq; mc_b = cdy_dy_r;  // k²·Δ(.y·D.y) → cap t=12
            end
            6'd13: begin                                                         // tick=14
                mc_a = b_cone_w; mc_b = b_cone_w;       // b² → cap t=15
            end
            6'd14: begin                                                         // tick=15
                mc_a = a_cone_r; mc_b = c_cone_r;       // a·c → cap t=16
            end
            default: begin end
        endcase
    end

    logic signed [WORD-1:0] b_cone_w, c_cone_w;
    assign b_cone_w = cdx_dx_r + cdz_dz_r - k_cdy_dy_r;
    assign c_cone_w = cdx_sq_r + cdz_sq_r - k_cdy_sq_r;

    // Two fp_sqrts in lockstep at tick=13.
    logic                  sqrt_start;
    logic signed [WORD-1:0] sqrt0_in, sqrt1_in;
    logic signed [WORD-1:0] sqrt0_out, sqrt1_out;
    logic                  sqrt0_done, sqrt1_done;
    assign sqrt_start = (tick == 6'd13);
    assign sqrt0_in   = half_b0_sq_r - a_c0_r;
```

```
    assign sqrt1_in   = half_b1_sq_r - a_c1_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt0 (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
        .a(sqrt0_in), .result(sqrt0_out), .done(sqrt0_done)
    );
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt1 (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
        .a(sqrt1_in), .result(sqrt1_out), .done(sqrt1_done)
    );

    // Cone disc sqrt
    logic                  sqrt_cone_start;
    logic signed [WORD-1:0] sqrt_cone_in;
    logic signed [WORD-1:0] sqrt_cone_out;
    logic                  sqrt_cone_done;
    assign sqrt_cone_start = (tick == 6'd17);
    assign sqrt_cone_in    = b_cone_sq_r - a_c_cone_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt_cone (
        .clk(clk), .rst_n(rst_n), .start(sqrt_cone_start),
        .a(sqrt_cone_in), .result(sqrt_cone_out), .done(sqrt_cone_done)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            dx_r <= '0; dy_r <= '0; dz_r <= '0;
            ox_r <= '0; oy_r <= '0; oz_r <= '0;
            a_r  <= '0;
            oc0_x_r <= '0; oc0_y_r <= '0; oc0_z_r <= '0;
            oc1_x_r <= '0; oc1_y_r <= '0; oc1_z_r <= '0;
            bag_r <= '0;
            p0_hb0_r <= '0; p1_hb0_r <= '0; p2_hb0_r <= '0;
            p0_oc0_r <= '0; p1_oc0_r <= '0; p2_oc0_r <= '0;
            half_b0_r <= '0; c0_r <= '0;
            half_b0_sq_r <= '0; a_c0_r <= '0;
            disc_sqrt0_r <= '0; no_hit0_r <= 1'b0;
            p0_hb1_r <= '0; p1_hb1_r <= '0; p2_hb1_r <= '0;
            p0_oc1_r <= '0; p1_oc1_r <= '0; p2_oc1_r <= '0;
            half_b1_r <= '0; c1_r <= '0;
            half_b1_sq_r <= '0; a_c1_r <= '0;
            disc_sqrt1_r <= '0; no_hit1_r <= 1'b0;
            cone_dx_r <= '0; cone_dy_r <= '0; cone_dz_r <= '0;
            cdx_sq_r <= '0; cdy_sq_r <= '0; cdz_sq_r <= '0; k_cdy_sq_r <= '0;
            cdx_dx_r <= '0; cdy_dy_r <= '0; cdz_dz_r <= '0; k_cdy_dy_r <= '0;
            a_cone_r <= '0; b_cone_r <= '0; c_cone_r <= '0;
            b_cone_sq_r <= '0; a_c_cone_r <= '0;
            disc_sqrt_cone_r <= '0; no_hit_cone_r <= 1'b0;
        end else begin
            case (tick)
                6'd0: begin
                    dx_r <= raygen_raw_x;
                    dy_r <= raygen_raw_y;
                    dz_r <= raygen_raw_z;
                    ox_r <= raygen_ox;
                    oy_r <= raygen_oy;
                    oz_r <= raygen_oz;
                    a_r  <= raygen_mag_sq;
                    a_cone_r <= raygen_a_cone;   // precomputed in stage A
                    bag_r <= raygen_bag;
                end
                6'd1: begin
                    oc0_x_r <= oc0_x_c; oc0_y_r <= oc0_y_c; oc0_z_r <= oc0_z_c;
                    oc1_x_r <= oc1_x_c; oc1_y_r <= oc1_y_c; oc1_z_r <= oc1_z_c;
                    cone_dx_r <= ox_r - in_scene.cone.apex_x;
                    cone_dy_r <= oy_r - in_scene.cone.apex_y;
                    cone_dz_r <= oz_r - in_scene.cone.apex_z;
                end
                6'd3: begin p0_hb0_r <= m0_r; p0_hb1_r <= m1_r; end
                6'd4: begin p1_hb0_r <= m0_r; p1_hb1_r <= m1_r; end
                6'd5: begin p2_hb0_r <= m0_r; p2_hb1_r <= m1_r; cdx_sq_r   <= mc_r; end
                6'd6: begin
                    p0_oc0_r  <= m0_r; p0_oc1_r  <= m1_r;
                    half_b0_r <= p0_hb0_r + p1_hb0_r + p2_hb0_r;
                    half_b1_r <= p0_hb1_r + p1_hb1_r + p2_hb1_r;
                    cdy_sq_r  <= mc_r;
```

```
                end
                6'd7: begin p1_oc0_r <= m0_r; p1_oc1_r <= m1_r; cdz_sq_r   <= mc_r; end
                6'd8: begin p2_oc0_r <= m0_r; p2_oc1_r <= m1_r; k_cdy_sq_r <= mc_r; end
                6'd9: begin
                    half_b0_sq_r <= m0_r; half_b1_sq_r <= m1_r;
                    c0_r <= p0_oc0_r + p1_oc0_r + p2_oc0_r - sc0_local.r_sq;
                    c1_r <= p0_oc1_r + p1_oc1_r + p2_oc1_r - sc1_local.r_sq;
                    cdx_dx_r <= mc_r;
                end
                6'd10: cdy_dy_r <= mc_r;
                6'd11: begin a_c0_r <= m0_r; a_c1_r <= m1_r; cdz_dz_r <= mc_r; end
                6'd12: begin
                    no_hit0_r <= (half_b0_sq_r < a_c0_r);
                    no_hit1_r <= (half_b1_sq_r < a_c1_r);
                    k_cdy_dy_r <= mc_r;
                end
                6'd14: begin
                    b_cone_r <= b_cone_w;
                    c_cone_r <= c_cone_w;
                end
                6'd15: b_cone_sq_r <= mc_r;
                6'd16: begin
                    a_c_cone_r    <= mc_r;
                    no_hit_cone_r <= (b_cone_sq_r < mc_r);
                end
                default: begin end
            endcase
            if (sqrt0_done) disc_sqrt0_r <= sqrt0_out;
            if (sqrt1_done) disc_sqrt1_r <= sqrt1_out;
            if (sqrt_cone_done) disc_sqrt_cone_r <= sqrt_cone_out;
        end
    end

    assign isect_dx = dx_r;
    assign isect_dy = dy_r;
    assign isect_dz = dz_r;
    assign isect_s0_nb      = -half_b0_r;
    assign isect_s0_dsq  = disc_sqrt0_r;
    assign isect_s0_nh     = no_hit0_r;
    assign isect_s1_nb      = -half_b1_r;
    assign isect_s1_dsq  = disc_sqrt1_r;
    assign isect_s1_nh     = no_hit1_r;
    // Clamp |dy|   1/2048 so plane t = -orig.y/dy can't blow up at horizon
    // rays (dy   0)
    localparam signed [WORD-1:0] PLANE_DENOM_EPS = 1 <<< (FRAC_BITS - 11);
    logic signed [WORD-1:0] plane_denom_c;
    assign plane_denom_c =
        (raygen_raw_y > -PLANE_DENOM_EPS && raygen_raw_y < PLANE_DENOM_EPS)
            ? ((raygen_raw_y >= 0) ? PLANE_DENOM_EPS : -PLANE_DENOM_EPS)
            : raygen_raw_y;

    // 2 concurrent fp_invs at t=0 emit 1/sphere_a and 1/plane_denom for
    // stage C's t-muls (replaces 3 fp_divs in the old split). Finishes by t=40.
    logic                   inv_start;
    assign inv_start = (tick == 6'd0);
    logic signed [WORD-1:0] inv_a_w, inv_pd_w;
    logic                   inv_a_done, inv_pd_done;
    fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_a (
        .clk(clk), .rst_n(rst_n), .start(inv_start),
        .b(raygen_mag_sq), .result(inv_a_w), .done(inv_a_done)
    );
    fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_pd (
        .clk(clk), .rst_n(rst_n), .start(inv_start),
        .b(plane_denom_c), .result(inv_pd_w), .done(inv_pd_done)
    );

    // Cone fp_inv kicks at tick=0 alongside u_inv_a/u_inv_pd. Input a_cone_r
    // is precomputed in stage A and latched here at t=0; result is wire-valid
    // during t=42 and captured by stage DE at its own t=0 boundary latch
    // (2 clock periods of →FFFF slack). See docs/cone_fp_inv_timing.html.
    logic                   inv_cone_start;
    assign inv_cone_start = (tick == 6'd0);
    logic signed [WORD-1:0] inv_a_cone_w;
    logic                   inv_a_cone_done;
```

```
    // Same-edge race fix: at tick=0 the boundary latch a_cone_r <=
    // raygen_a_cone fires on the SAME posedge that fp_inv samples b.
    // fp_inv would see the pre-edge a_cone_r (one II behind). Feed the
    // upstream wire instead - it's been stable since stage A's t=13 of
    // the previous II.
    fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_a_cone (
        .clk(clk), .rst_n(rst_n), .start(inv_cone_start),
        .b(raygen_a_cone), .result(inv_a_cone_w), .done(inv_a_cone_done)
    );

    assign isect_inv_sa = inv_a_w;
    assign isect_inv_pd = inv_pd_w;
    assign isect_sa     = a_r;
    assign isect_inv_a_cone = inv_a_cone_w;
    assign isect_cone_nb    = -b_cone_r;
    assign isect_cone_dsq   = disc_sqrt_cone_r;
    assign isect_cone_nh    = no_hit_cone_r;
    assign isect_cone_dx    = cone_dx_r;
    assign isect_cone_dy    = cone_dy_r;
    assign isect_cone_dz    = cone_dz_r;

    assign isect_ox             = ox_r;
    assign isect_oy             = oy_r;
    assign isect_oz             = oz_r;
    assign isect_bag            = bag_r;
endmodule

// Stage C - HIT RESOLVE: intersect t-muls + winner mux (first half) fused
// with hit point + base color + normal prep (second half). Long pole is the
// diff_mag sqrt, completing by t=34. hit_type 3'b000=sky / 001=s0 / 010=s1 /
// 011=plane / 100=cone.
module rtl_hit_resolve
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                                 // clock, async neg-edge reset
    input  logic [5:0]              tick,                                       // pipe-local tick counter
    input  logic signed [WORD-1:0]  isect_dx, isect_dy, isect_dz,               // ray direction (from Stage B)
    input  logic signed [WORD-1:0]  isect_ox, isect_oy, isect_oz,               // ray origin
    input  logic signed [WORD-1:0]  isect_inv_sa,                               // 1/|d|² for sphere t-solve
    input  logic signed [WORD-1:0]  isect_s0_nb,                                // sphere 0: -half_b
    input  logic signed [WORD-1:0]  isect_s0_dsq,                               // sphere 0: √disc
    input  logic                    isect_s0_nh,                                // sphere 0: no_hit flag
    input  logic signed [WORD-1:0]  isect_s1_nb,                                // sphere 1: -half_b
    input  logic signed [WORD-1:0]  isect_s1_dsq,                               // sphere 1: √disc
    input  logic                    isect_s1_nh,                                // sphere 1: no_hit flag
    input  logic signed [WORD-1:0]  isect_inv_pd,                               // 1/dir.y for plane t-solve
    input  logic signed [WORD-1:0]  isect_sa,                                   // |d|² for sun squared-compare
    // Cone path inputs from rtl_isect_prep. inv_a_cone is captured later
    // (tick=14) since C's fp_inv kicked at C tick=15; the rest are stable
    // from start of II.
    input  logic signed [WORD-1:0]  isect_cone_nb,                              // cone: -b_cone
    input  logic signed [WORD-1:0]  isect_cone_dsq,                             // cone: √disc
    input  logic                    isect_cone_nh,                              // cone: no_hit flag
    input  logic signed [WORD-1:0]  isect_inv_a_cone,                           // 1/a_cone (late settle, captured at
 tick=14)
    input  logic signed [WORD-1:0]  isect_cone_dx,                              // cone Δ.x = orig.x - apex.x
    input  logic signed [WORD-1:0]  isect_cone_dy,                              // cone Δ.y (for slab + cap)
    input  logic signed [WORD-1:0]  isect_cone_dz,                              // cone Δ.z
    input  logic signed [WORD-1:0]  light_x_local, light_y_local, light_z_local, // light pos (per-pipe relay) for
sun + Lambert prep
    input  scene_t                  in_scene,                                   // scene relay (this stage reads
every field)
    input  logic signed [WORD-1:0]  K_const_local,                              // |light|²·63/64 - sun-billboard
threshold
    input  pipe_baggage_t           isect_bag,                                  // {sv, tag, meta} from Stage B
    output logic signed [WORD-1:0]  hit_hx, hit_hy, hit_hz,                     // surface hit point
    output logic signed [WORD-1:0]  hit_dfx, hit_dfy, hit_dfz,                  // diff = hit - sphere_center (sphere
 normal num)
    output logic signed [WORD-1:0]  hit_tlx, hit_tly, hit_tlz,                  // to-light vector tl = light - hit
```

```
    output logic signed [WORD-1:0]  hit_cr, hit_cg, hit_cb,                  // base surface RGB (after checker/sun mux)
    output logic [2:0]              hit_hit_type,                            // 000 sky, 001 s0, 010 s1, 011 plane, 100 cone
    output logic signed [WORD-1:0]  hit_dmag,                                // |diff| for sphere - needed by Stage D's 1/|n|
    // Forward parent direction to stage E (for refl r = d - 2·(d·n)·n).
    output logic signed [WORD-1:0]  hit_dx, hit_dy, hit_dz,                  // parent ray direction → Stage E refl formula
    output pipe_baggage_t           hit_bag                                  // baggage forwarded to Stage D
);
    localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
    localparam signed [WORD-1:0] SKY2_G = (7 * ONE) / 10;
    localparam signed [WORD-1:0] SUN_COL_R = ONE;                    // 1.0
    localparam signed [WORD-1:0] SUN_COL_G = (95 * ONE) / 100;       // 0.95
    localparam signed [WORD-1:0] SUN_COL_B = (55 * ONE) / 100;       // 0.55

    // ---- Tick=0 latches: dir, orig, light, sphere data, tag, meta ----
    logic signed [WORD-1:0] dx_r, dy_r, dz_r;
    logic signed [WORD-1:0] ox_r, oy_r, oz_r;
    logic signed [WORD-1:0] lx_r, ly_r, lz_r;

    logic signed [WORD-1:0] a_r;
    logic                   no_hit0_r, no_hit1_r;
    pipe_baggage_t          bag_r;

    // ---- ISECT (first half of stage C - intersect t-muls + winner mux) ----
    // Combinational numerators from upstream wires (race-fix).
    logic signed [WORD-1:0] num_t0_s0, num_t0_s1;
    assign num_t0_s0 = isect_s0_nb - isect_s0_dsq;
    assign num_t0_s1 = isect_s1_nb - isect_s1_dsq;
    logic signed [WORD-1:0] plane_numer_w;
    assign plane_numer_w = -isect_oy;

    logic signed [WORD-1:0] num_t0_s0_r, num_t0_s1_r, plane_numer_r;
    logic signed [WORD-1:0] inv_a_r, inv_pd_r;

    // t-mul outputs (registered after DSP).
    logic signed [WORD-1:0] t0_s0_r, t0_s1_r, tp_r;

    // Winner mux output, then registered to break C-side comb chain off
    // the hit-mul DSP a-input.
    logic signed [WORD-1:0] hit_t_w, hit_t_r;
    logic [2:0]             hit_type_w, hit_type_r;

    // ---- HP + NORMAL PREP (second half of stage C) ----
    logic signed [WORD-1:0] m_dh_x_r, m_dh_y_r, m_dh_z_r;
    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
    logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] sky_g_r;
    logic signed [WORD-1:0] prod0_r, prod1_r, prod2_r;
    logic signed [WORD-1:0] diff_mag_r;
    // Pipelined t_sky / inv_t_sky to lift col_*_r off the dy_r adder cascade.
    logic signed [WORD-1:0] t_sky_r, inv_t_sky_r;

    // ---- Cone state (parallel to sphere/plane path) ----
    logic signed [WORD-1:0] cone_nb_r, cone_dsq_r;
    logic                   cone_nh_r;
    logic signed [WORD-1:0] cone_dx_r, cone_dy_r, cone_dz_r;       // Δ (origin - apex)
    logic signed [WORD-1:0] inv_a_cone_r;
    logic signed [WORD-1:0] t_cone_r;
    logic signed [WORD-1:0] m_dh_cx_r, m_dh_cy_r, m_dh_cz_r;       // dir·t_cone
    logic signed [WORD-1:0] hit_x_cone_r, hit_y_cone_r, hit_z_cone_r;
    logic                   cone_in_slab_r;                        // hit_y in [apex.y-h, apex.y]
    logic                   cone_wins_r;                           // final mux decision
    logic signed [WORD-1:0] cone_norm_x_r, cone_norm_y_r, cone_norm_z_r; // unnormalized gradient
    logic signed [WORD-1:0] cone_kdy_r;                            // k²·(hit_y-apex_y) pre-neg
    logic signed [WORD-1:0] cone_diff_mag_r;                       // factor · (apex.y - hit.y)

    // Single shared fp_mul scheduled across t-muls, hit-muls, sky, diff².
    logic signed [WORD-1:0] mul_a, mul_b, mul_result;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (
```

```
        .clk(clk), .a(mul_a), .b(mul_b), .result(mul_result)
    );

    logic signed [WORD-1:0] mc_a, mc_b, mc_r;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone_e (
        .clk(clk), .a(mc_a), .b(mc_b), .result(mc_r)
    );

    logic signed [WORD-1:0] sun_x_r, sun_y_r, sun_z_r;
    logic signed [WORD-1:0] sun_dot_c;
    logic signed [WORD-1:0] sun_dot_r;
    assign sun_dot_c = sun_x_r + sun_y_r + sun_z_r;

    // Squared-compare regs (u_mul time-mux'd at t=9/10): K_const·a_r →
    // sun_thresh_a_r; sun_dot²; combinational sign+GE; 1c FF for safety.
    logic signed [WORD-1:0] sun_thresh_a_r;
    logic signed [WORD-1:0] sun_dot_sq_r;
    logic                   sun_active_w;
    // Sign gate (~sun_dot_r MSB) catches antipodal rays whose squared
    // compare would otherwise pass.
    assign sun_active_w = (~sun_dot_r[WORD-1])
                      && (sun_dot_sq_r >= sun_thresh_a_r);
    logic sun_active_r;

    // sky t = (dir_y + ONE) >>> 1 (combinational from registered dy_r).
    logic signed [WORD-1:0] t_sky_c;
    assign t_sky_c = (dy_r + ONE) >>> 1;

    // Locally-registered `tick` for the operand muxes. `tick` has ~400
    // destinations across the pipe, and the cross-die IC from tick[3] to
    // the cone-mul DSP's selector cluster was the worst-slack path
    // (-0.477 ns @ SEED=2). This single FF lags `tick` by 1 cycle and
    // drives BOTH operand muxes (u_mul and u_mul_cone_e); no preserve
    // attribute so Quartus is free to place / replicate it as needed near
    // the consumer DSPs. Same pattern as the validated stage_e fix that
    // closed -0.091 ns at the visual-upgrades baseline. Case arm indices
    // shift by -1; always_ff captures still use `tick` directly so the
    // tick=0 input latch and downstream stage handshake are unchanged.
    logic [5:0] tick_local_h;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) tick_local_h <= 6'd0;
        else        tick_local_h <= tick;
    end

    always_comb begin
        mul_a = '0; mul_b = '0;
        case (tick_local_h)
            6'd0:  begin mul_a = num_t0_s0_r;   mul_b = inv_a_r;   end  // tick=1
            6'd1:  begin mul_a = num_t0_s1_r;   mul_b = inv_a_r;   end  // tick=2
            6'd2:  begin mul_a = plane_numer_r; mul_b = inv_pd_r;  end  // tick=3
            6'd3:  begin mul_a = SKY2_G;        mul_b = t_sky_r;   end  // tick=4
            6'd4:  begin mul_a = dx_r;          mul_b = lx_r;      end  // tick=5
            6'd5:  begin mul_a = dy_r;          mul_b = ly_r;      end  // tick=6
            6'd6:  begin mul_a = dz_r;          mul_b = lz_r;      end  // tick=7
            6'd8:  begin mul_a = K_const_local; mul_b = a_r;       end  // tick=9
            6'd9:  begin mul_a = dx_r;          mul_b = hit_t_r;   end  // tick=10
            6'd10: begin mul_a = dy_r;          mul_b = hit_t_r;   end  // tick=11
            6'd11: begin mul_a = dz_r;          mul_b = hit_t_r;   end  // tick=12
            6'd12: begin mul_a = sun_dot_r;     mul_b = sun_dot_r; end  // tick=13
            6'd15: begin mul_a = diff_x_r;      mul_b = diff_x_r;  end  // tick=16
            6'd16: begin mul_a = diff_y_r;      mul_b = diff_y_r;  end  // tick=17
            6'd17: begin mul_a = diff_z_r;      mul_b = diff_z_r;  end  // tick=18
            default: begin end
        endcase
    end

    // Cone fp_mul scheduler. inv_a_cone_r is now captured at t=0 (boundary
    // latch alongside inv_a_r/inv_pd_r), so the t=15 mux reads a settled
    // value with plenty of slack.
    logic signed [WORD-1:0] num_cone_w;
    assign num_cone_w = cone_nb_r - cone_dsq_r;  // = -b - sqrt_disc

    // Apex cull: reject cone hits within CONE_APEX_CULL of the apex
    // (1/32 m at FRAC_BITS=13)
```

```
    localparam signed [WORD-1:0] CONE_APEX_CULL = 1 <<< (FRAC_BITS - 5);

    always_comb begin
        mc_a = '0; mc_b = '0;
        case (tick_local_h)
            6'd14: begin mc_a = num_cone_w;     mc_b = inv_a_cone_r; end                       // tick=15
            6'd16: begin mc_a = dx_r;           mc_b = t_cone_r;     end                       // tick=17
            6'd17: begin mc_a = dy_r;           mc_b = t_cone_r;     end                       // tick=18
            6'd18: begin mc_a = dz_r;           mc_b = t_cone_r;     end                       // tick=19
            6'd21: begin mc_a = in_scene.cone.k_sq;                                            // tick=22
                         mc_b = (hit_y_cone_r - in_scene.cone.apex_y); end
            6'd22: begin mc_a = in_scene.cone.norm_factor;                                     // tick=23
                         mc_b = (in_scene.cone.apex_y - hit_y_cone_r); end
            default: begin end
        endcase
    end

    // ---- Per-candidate hit flags (read registered t values) ----
    logic s0_hit, s1_hit, plane_hit;
    assign s0_hit    = !no_hit0_r && (t0_s0_r > 0);
    assign s1_hit    = !no_hit1_r && (t0_s1_r > 0);
    assign plane_hit = (tp_r > 0);

    // Closest-t winner (3 parallel pairwise compares + win-flag AND + 4-way
    // mux, ~5 LUT levels). Tie-breaking: s0 > s1 > plane.
    logic s1_lt_s0, tp_lt_s0, tp_lt_s1;
    assign s1_lt_s0 = t0_s1_r < t0_s0_r;
    assign tp_lt_s0 = tp_r    < t0_s0_r;
    assign tp_lt_s1 = tp_r    < t0_s1_r;
    logic s0_wins, s1_wins, plane_wins;
    assign s0_wins    = s0_hit    && (!s1_hit || !s1_lt_s0) && (!plane_hit || !tp_lt_s0);
    assign s1_wins    = s1_hit    && (!s0_hit ||  s1_lt_s0) && (!plane_hit || !tp_lt_s1);
    assign plane_wins = plane_hit && (!s0_hit ||  tp_lt_s0) && (!s1_hit    ||  tp_lt_s1);
    always_comb begin
        unique case ({s0_wins, s1_wins, plane_wins})
            3'b100:  begin hit_t_w = t0_s0_r; hit_type_w = HIT_TYPE_S0;    end
            3'b010:  begin hit_t_w = t0_s1_r; hit_type_w = HIT_TYPE_S1;    end
            3'b001:  begin hit_t_w = tp_r;    hit_type_w = HIT_TYPE_PLANE; end
            default: begin hit_t_w = '0;      hit_type_w = HIT_TYPE_SKY;   end
        endcase
    end

    // Floor pattern from hit_x_r/hit_z_r bits. Modes: 00=1m checker,
    // 01=Sierpinski (3-bit AND==0 on 8m tile), 10=diagonal stripes,
    // 11=0.5m checker. Two's-complement bit extract works across origin.
    logic chk_bit;
    always_comb begin
        unique case (in_scene.floor.mode)
            FLOOR_MODE_1M_CHECK:   chk_bit = hit_x_r[FRAC_BITS]   ^ hit_z_r[FRAC_BITS];
            FLOOR_MODE_SIERPINSKI: chk_bit = (hit_x_r[FRAC_BITS+:3] & hit_z_r[FRAC_BITS+:3]) == 3'b000;
            FLOOR_MODE_DIAGONAL:   chk_bit = hit_x_r[FRAC_BITS+1] ^ hit_z_r[FRAC_BITS];
            FLOOR_MODE_FINE_CHECK: chk_bit = hit_x_r[FRAC_BITS-1] ^ hit_z_r[FRAC_BITS-1];
            default:               chk_bit = hit_x_r[FRAC_BITS]   ^ hit_z_r[FRAC_BITS];
        endcase
    end

    logic chk_bit_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) chk_bit_r <= 1'b0;
        else        chk_bit_r <= chk_bit;
    end

    function automatic logic signed [WORD-1:0] clamp01(input logic signed [WORD-1:0] v);
        if (v < 0)        return '0;
        else if (v > ONE) return ONE;
        else              return v;
    endfunction

    // ---- Sphere center select (registered hit_type_r drives the mux) ----
    logic signed [WORD-1:0] sc_sel_x, sc_sel_y, sc_sel_z;
    assign sc_sel_x = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.x : in_scene.s0.x;
    assign sc_sel_y = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.y : in_scene.s0.y;
    assign sc_sel_z = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.z : in_scene.s0.z;

```

```
    logic signed [WORD-1:0] sc_sel_x_r, sc_sel_y_r, sc_sel_z_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            sc_sel_x_r <= '0;
            sc_sel_y_r <= '0;
            sc_sel_z_r <= '0;
        end else begin
            sc_sel_x_r <= sc_sel_x;
            sc_sel_y_r <= sc_sel_y;
            sc_sel_z_r <= sc_sel_z;
        end
    end

    // Sky-gradient combinationals
    logic signed [WORD-1:0] inv_t_sky_c;
    logic signed [WORD-1:0] sky_r_c, sky_g_full_c, sky_b_c;
    assign inv_t_sky_c   = ONE - t_sky_c;
    assign sky_r_c       = clamp01(inv_t_sky_r + (t_sky_r >>> 1));
    assign sky_g_full_c  = clamp01(inv_t_sky_r + sky_g_r);
    assign sky_b_c       = ONE;

    // Base color mux on hit_type: 000=sky, 001=s0, 010=s1, 011=plane (chk_bit), 100=cone.
    logic signed [WORD-1:0] col_r_c, col_g_c, col_b_c;
    always_comb begin
        case (hit_type_r)
            HIT_TYPE_S0: begin
                col_r_c = in_scene.s0.color.r; col_g_c = in_scene.s0.color.g; col_b_c = in_scene.s0.color.b;
            end
            HIT_TYPE_S1: begin
                col_r_c = in_scene.s1.color.r; col_g_c = in_scene.s1.color.g; col_b_c = in_scene.s1.color.b;
            end
            HIT_TYPE_PLANE: begin
                if (!chk_bit_r) begin
                    col_r_c = in_scene.floor.pc1.r; col_g_c = in_scene.floor.pc1.g; col_b_c = in_scene.floor.pc1.b;
                end else begin
                    col_r_c = in_scene.floor.pc2.r; col_g_c = in_scene.floor.pc2.g; col_b_c = in_scene.floor.pc2.b;
                end
            end
            default: begin
                col_r_c = sky_r_c;
                col_g_c = sky_g_full_c;
                col_b_c = sky_b_c;
            end
        endcase
    end

    logic                   sqrt_start;
    logic signed [WORD-1:0] sqrt_in;
    logic signed [WORD-1:0] sqrt_out;
    logic                   sqrt_done;
    assign sqrt_start = (tick == 6'd20);
    assign sqrt_in    = prod0_r + prod1_r + prod2_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
        .a(sqrt_in), .result(sqrt_out), .done(sqrt_done)
    );

    // ---- All FFs ----
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            dx_r <= '0; dy_r <= '0; dz_r <= '0;
            ox_r <= '0; oy_r <= '0; oz_r <= '0;
            lx_r <= '0; ly_r <= '0; lz_r <= '0;
            a_r  <= '0;
            no_hit0_r <= 1'b0; no_hit1_r <= 1'b0;
            bag_r <= '0;
            num_t0_s0_r <= '0; num_t0_s1_r <= '0; plane_numer_r <= '0;
            inv_a_r <= '0; inv_pd_r <= '0;
            t0_s0_r <= '0; t0_s1_r <= '0; tp_r <= '0;
            hit_t_r <= '0; hit_type_r <= HIT_TYPE_SKY;
            m_dh_x_r <= '0; m_dh_y_r <= '0; m_dh_z_r <= '0;
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
            tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
```

```
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            sky_g_r <= '0;
            prod0_r <= '0; prod1_r <= '0; prod2_r <= '0;
            diff_mag_r <= '0;
            t_sky_r <= '0; inv_t_sky_r <= '0;
            sun_thresh_a_r <= '0;
            sun_dot_sq_r <= '0;
            sun_dot_r <= '0;
            sun_active_r <= 1'b0;
            sun_x_r <= '0; sun_y_r <= '0; sun_z_r <= '0;
            cone_nb_r <= '0; cone_dsq_r <= '0; cone_nh_r <= 1'b0;
            cone_dx_r <= '0; cone_dy_r <= '0; cone_dz_r <= '0;
            inv_a_cone_r <= '0;
            t_cone_r <= '0;
            m_dh_cx_r <= '0; m_dh_cy_r <= '0; m_dh_cz_r <= '0;
            hit_x_cone_r <= '0; hit_y_cone_r <= '0; hit_z_cone_r <= '0;
            cone_in_slab_r <= 1'b0;
            cone_wins_r <= 1'b0;
            cone_norm_x_r <= '0; cone_norm_y_r <= '0; cone_norm_z_r <= '0;
            cone_kdy_r <= '0;
            cone_diff_mag_r <= '0;
        end else begin
            t_sky_r     <= t_sky_c;
            inv_t_sky_r <= inv_t_sky_c;
            case (tick)
                6'd0: begin
                    dx_r <= isect_dx; dy_r <= isect_dy; dz_r <= isect_dz;
                    ox_r <= isect_ox; oy_r <= isect_oy; oz_r <= isect_oz;
                    lx_r <= light_x_local; ly_r <= light_y_local; lz_r <= light_z_local;
                    a_r  <= isect_sa;
                    no_hit0_r <= isect_s0_nh || isect_bag.meta.excluded[0];
                    no_hit1_r <= isect_s1_nh || isect_bag.meta.excluded[1];
                    bag_r <= isect_bag;
                    num_t0_s0_r   <= num_t0_s0;
                    num_t0_s1_r   <= num_t0_s1;
                    plane_numer_r <= plane_numer_w;
                    inv_a_r       <= isect_inv_sa;
                    inv_pd_r      <= isect_inv_pd;
                    // Cone fp_inv result is captured here at the boundary
                    // latch alongside inv_a_r/inv_pd_r (cone kicks at C-t=0
                    // now that a_cone is precomputed in stage A).
                    inv_a_cone_r  <= isect_inv_a_cone;
                    // Cone Stage C outputs (stable from start of II).
                    // excluded[2] is the cone self-skip bit on reflection rays.
                    cone_nb_r  <= isect_cone_nb;
                    cone_dsq_r <= isect_cone_dsq;
                    cone_nh_r  <= isect_cone_nh || isect_bag.meta.excluded[2];
                    cone_dx_r  <= isect_cone_dx;
                    cone_dy_r  <= isect_cone_dy;
                    cone_dz_r  <= isect_cone_dz;
                end
                6'd2: t0_s0_r <= mul_result;
                6'd3: t0_s1_r <= mul_result;
                6'd4: tp_r    <= mul_result;
                6'd5: begin
                    hit_t_r    <= hit_t_w;
                    hit_type_r <= hit_type_w;
                    sky_g_r    <= mul_result;
                end
                6'd6: sun_x_r <= mul_result;
                6'd7: sun_y_r <= mul_result;
                6'd8: sun_z_r <= mul_result;
                6'd9: sun_dot_r <= sun_dot_c;
                6'd10: sun_thresh_a_r <= mul_result;
                6'd11: m_dh_x_r <= mul_result;               // HP-x from t=10 issue
                6'd12: m_dh_y_r <= mul_result;               // HP-y from t=11
                6'd13: m_dh_z_r <= mul_result;               // HP-z from t=12
                6'd14: begin
                    hit_x_r <= ox_r + m_dh_x_r;
                    hit_y_r <= oy_r + m_dh_y_r;
                    hit_z_r <= oz_r + m_dh_z_r;
                    sun_dot_sq_r <= mul_result;              // sun² from t=13 issue
                end
                6'd15: begin
```

```
                diff_x_r <= hit_x_r - sc_sel_x_r;
                diff_y_r <= hit_y_r - sc_sel_y_r;
                diff_z_r <= hit_z_r - sc_sel_z_r;
                tl_x_r   <= lx_r - hit_x_r;
                tl_y_r   <= ly_r - hit_y_r;
                tl_z_r   <= lz_r - hit_z_r;
                sun_active_r <= sun_active_w;
            end
            6'd16: begin
                if (hit_type_r == HIT_TYPE_SKY && sun_active_r) begin
                    col_r_r <= SUN_COL_R;
                    col_g_r <= SUN_COL_G;
                    col_b_r <= SUN_COL_B;
                end else begin
                    col_r_r <= col_r_c;
                    col_g_r <= col_g_c;
                    col_b_r <= col_b_c;
                end
                t_cone_r <= mc_r;                       // = num_cone · inv_a_cone
            end
            6'd17: prod0_r <= mul_result;               // diff_x² from t=16
            6'd18: begin
                prod1_r   <= mul_result;                // diff_y² from t=17
                m_dh_cx_r <= mc_r;                      // from t=17 mul (dx · t_cone_r settled)
            end
            6'd19: begin
                prod2_r   <= mul_result;                // diff_z² from t=18
                m_dh_cy_r <= mc_r;                      // from t=18
            end
            6'd20: m_dh_cz_r <= mc_r;                   // from t=19
            6'd21: begin
                // Cone hit point. m_dh_c*_r are all settled by now.
                hit_x_cone_r <= ox_r + m_dh_cx_r;
                hit_y_cone_r <= oy_r + m_dh_cy_r;
                hit_z_cone_r <= oz_r + m_dh_cz_r;
            end
            6'd23: begin
                cone_kdy_r     <= mc_r;                 // from t=22 mul
                // Slab boundary + apex cull. Uses settled hit_y_cone_r
                // (registered at t=21, settled t=22, read at t=23 LHS).
                cone_in_slab_r <= ( (hit_y_cone_r <= (in_scene.cone.apex_y - CONE_APEX_CULL)) &&
                                    (hit_y_cone_r >= (in_scene.cone.apex_y - in_scene.cone.height)) );
                // Surface gradient X/Z components (Y handled at t=24
                // because it needs the freshly-captured cone_kdy_r).
                cone_norm_x_r <= hit_x_cone_r - in_scene.cone.apex_x;
                cone_norm_z_r <= hit_z_cone_r - in_scene.cone.apex_z;
            end
            6'd24: begin
                cone_wins_r     <= !cone_nh_r && (t_cone_r > 0) && cone_in_slab_r &&
                                   ((hit_type_r == HIT_TYPE_SKY) ||
                                    (t_cone_r < hit_t_r));
                // Y component of gradient = -k²·(hit_y - apex_y); cone_kdy_r
                // holds k²·(hit_y - apex_y) with sign so we just negate
                cone_norm_y_r   <= -cone_kdy_r;
                cone_diff_mag_r <= mc_r;                // from t=23 mul: factor · (apex_y - hit_y)
            end
            default: begin end
        endcase
        if (sqrt_done) diff_mag_r <= sqrt_out;
    end
end

// ---- Output muxing: cone_wins_r overrides hit_*_r and diff vector ----
logic [2:0]              final_hit_type;
logic signed [WORD-1:0]  final_hx, final_hy, final_hz;
logic signed [WORD-1:0]  final_dfx, final_dfy, final_dfz;
logic signed [WORD-1:0]  final_dmag;
logic signed [WORD-1:0]  final_cr, final_cg, final_cb;
logic signed [WORD-1:0]  final_tlx, final_tly, final_tlz;

always_comb begin
    if (cone_wins_r) begin
        final_hit_type = HIT_TYPE_CONE;
        final_hx       = hit_x_cone_r;
```

```
            final_hy       = hit_y_cone_r;
            final_hz       = hit_z_cone_r;
            final_dfx      = cone_norm_x_r;
            final_dfy      = cone_norm_y_r;
            final_dfz      = cone_norm_z_r;
            final_dmag     = cone_diff_mag_r;
            final_cr       = in_scene.cone.color.r;
            final_cg       = in_scene.cone.color.g;
            final_cb       = in_scene.cone.color.b;
            final_tlx      = lx_r - hit_x_cone_r;
            final_tly      = ly_r - hit_y_cone_r;
            final_tlz      = lz_r - hit_z_cone_r;
        end else begin
            final_hit_type = hit_type_r;
            final_hx       = hit_x_r;
            final_hy       = hit_y_r;
            final_hz       = hit_z_r;
            final_dfx      = diff_x_r;
            final_dfy      = diff_y_r;
            final_dfz      = diff_z_r;
            final_dmag     = diff_mag_r;
            final_cr       = col_r_r;
            final_cg       = col_g_r;
            final_cb       = col_b_r;
            final_tlx      = tl_x_r;
            final_tly      = tl_y_r;
            final_tlz      = tl_z_r;
        end
    end

    assign hit_hx = final_hx;
    assign hit_hy = final_hy;
    assign hit_hz = final_hz;
    assign hit_dfx = final_dfx;
    assign hit_dfy = final_dfy;
    assign hit_dfz = final_dfz;
    assign hit_tlx = final_tlx;
    assign hit_tly = final_tly;
    assign hit_tlz = final_tlz;
    assign hit_cr = final_cr;
    assign hit_cg = final_cg;
    assign hit_cb = final_cb;
    assign hit_hit_type = final_hit_type;
    assign hit_dmag = final_dmag;
    assign hit_dx = dx_r;
    assign hit_dy = dy_r;
    assign hit_dz = dz_r;
    assign hit_bag = bag_r;
endmodule

// Stage D - NORMAL INV: emit 1/diff_mag (single fp_inv) + ride-along forward
module rtl_normal_div
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                      // clock, async neg-edge reset
    input  logic [5:0]              tick,                             // pipe-local tick counter
    input  logic signed [WORD-1:0]  hit_hx, hit_hy, hit_hz,           // surface hit point (forwarded)
    input  logic signed [WORD-1:0]  hit_dfx, hit_dfy, hit_dfz,        // diff = hit - sphere_center (sphere normal
 numerator)
    input  logic signed [WORD-1:0]  hit_tlx, hit_tly, hit_tlz,        // to-light vector (forwarded)
    input  logic signed [WORD-1:0]  hit_cr, hit_cg, hit_cb,           // base surface RGB (forwarded)
    input  logic [2:0]              hit_hit_type,                     // hit type (forwarded)
    input  logic signed [WORD-1:0]  hit_dmag,                         // |diff| - sqrt result from Stage C
    input  logic signed [WORD-1:0]  hit_dx, hit_dy, hit_dz,           // parent ray direction (forwarded to Stage
E)
    input  pipe_baggage_t           hit_bag,                          // baggage
    output logic signed [WORD-1:0]  norm_hx, norm_hy, norm_hz,        // hit point (forwarded)
    output logic signed [WORD-1:0]  norm_inv_dm,                      // 1/|diff| - sphere normal magnitude
inverse (THIS STAGE'S WORK)
```

```
    output logic signed [WORD-1:0]  norm_dfx, norm_dfy, norm_dfz,      // diff (forwarded to Stage E for n = diff·inv_dm)
    output logic signed [WORD-1:0]  norm_tlx, norm_tly, norm_tlz,      // to-light vector (forwarded)
    output logic signed [WORD-1:0]  norm_cr, norm_cg, norm_cb,         // base color (forwarded)
    output logic [2:0]              norm_hit_type,                     // hit type (forwarded)
    output logic signed [WORD-1:0]  norm_dx, norm_dy, norm_dz,         // ray direction (forwarded)
    output pipe_baggage_t           norm_bag                           // baggage forwarded to Stage E
);
    localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;

    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
    logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] dx_r, dy_r, dz_r;
    logic [2:0]             hit_type_r;
    pipe_baggage_t          bag_r;

    logic                   inv_start;
    assign inv_start = (tick == 6'd0);
    logic signed [WORD-1:0] inv_dm_w;
    logic                   inv_dm_done;
    fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_dm (
        .clk(clk), .rst_n(rst_n), .start(inv_start),
        .b(hit_dmag), .result(inv_dm_w), .done(inv_dm_done)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
            tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            dx_r <= '0; dy_r <= '0; dz_r <= '0;
            hit_type_r <= HIT_TYPE_SKY;
            bag_r <= '0;
        end else if (tick == 6'd0) begin
            hit_x_r <= hit_hx; hit_y_r <= hit_hy; hit_z_r <= hit_hz;
            diff_x_r <= hit_dfx; diff_y_r <= hit_dfy; diff_z_r <= hit_dfz;
            tl_x_r <= hit_tlx; tl_y_r <= hit_tly; tl_z_r <= hit_tlz;
            col_r_r <= hit_cr; col_g_r <= hit_cg; col_b_r <= hit_cb;
            dx_r <= hit_dx; dy_r <= hit_dy; dz_r <= hit_dz;
            hit_type_r <= hit_hit_type;
            bag_r <= hit_bag;
        end
    end

    assign norm_hx = hit_x_r;
    assign norm_hy = hit_y_r;
    assign norm_hz = hit_z_r;
    assign norm_inv_dm = inv_dm_w;
    assign norm_dfx = diff_x_r;
    assign norm_dfy = diff_y_r;
    assign norm_dfz = diff_z_r;
    assign norm_tlx = tl_x_r;
    assign norm_tly = tl_y_r;
    assign norm_tlz = tl_z_r;
    assign norm_cr = col_r_r;
    assign norm_cg = col_g_r;
    assign norm_cb = col_b_r;
    assign norm_hit_type = hit_type_r;
    assign norm_dx = dx_r;
    assign norm_dy = dy_r;
    assign norm_dz = dz_r;
    assign norm_bag = bag_r;
endmodule

// Stage E - REFLECT DIR + LIGHT PREP: recovers normal via fp_mul (diff·1/dm),
// computes refl = d - 2(d·n)n, and the light_dist sqrt for stage F
module rtl_reflect_dir
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
```

```
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                        // clock, async neg-edge reset
    input  logic [5:0]              tick,                               // pipe-local tick counter
    input  logic signed [WORD-1:0]  norm_hx, norm_hy, norm_hz,          // surface hit point (forwarded)
    input  logic signed [WORD-1:0]  norm_inv_dm,                        // 1/|diff| from Stage D (used for sphere n
= diff·inv_dm)
    input  logic signed [WORD-1:0]  norm_dfx, norm_dfy, norm_dfz,       // diff = hit - center (for sphere normal)
    input  logic signed [WORD-1:0]  norm_tlx, norm_tly, norm_tlz,       // to-light vector (forwarded; |tl|² used
for sqrt)
    input  logic signed [WORD-1:0]  norm_cr, norm_cg, norm_cb,          // base color (forwarded)
    input  logic [2:0]              norm_hit_type,                      // hit type (plane override → n = (0,1,0))
    input  logic signed [WORD-1:0]  norm_dx, norm_dy, norm_dz,          // parent ray direction (for reflection
formula)
    input  pipe_baggage_t           norm_bag,                           // baggage
    output logic signed [WORD-1:0]  refl_hx, refl_hy, refl_hz,          // hit point (forwarded)
    output logic signed [WORD-1:0]  refl_nx, refl_ny, refl_nz,          // unit normal n (computed here for spheres;
 (0,1,0) for plane)
    output logic signed [WORD-1:0]  refl_tlx, refl_tly, refl_tlz,       // to-light vector (forwarded)
    output logic signed [WORD-1:0]  refl_cr, refl_cg, refl_cb,          // base color (forwarded)
    output logic [2:0]              refl_hit_type,                      // hit type (forwarded)
    output logic signed [WORD-1:0]  refl_ld,                            // |to-light| = √(tl·tl) - light distance (
sqrt computed here)
    // refl = d - 2·(d·n)·n, forwarded through H/IJ to K's spawn router.
    output logic signed [WORD-1:0]  refl_rx, refl_ry, refl_rz,          // reflection vector r - child ray direction
 if spawn
    output pipe_baggage_t           refl_bag                            // baggage forwarded to Stage F
);
    localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;

    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] inv_dm_r;
    logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
    logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] dx_r, dy_r, dz_r;
    logic [2:0]             hit_type_r;
    pipe_baggage_t          bag_r;

    // Surface-normal components from the t3..5 diff·inv_dm muls.
    logic signed [WORD-1:0] mul_nx_r, mul_ny_r, mul_nz_r;

    // Plane override: hit_type==HIT_TYPE_PLANE forces n=(0,1,0).
    logic signed [WORD-1:0] norm_x_w, norm_y_w, norm_z_w;
    always_comb begin
        if (hit_type_r == HIT_TYPE_PLANE) begin
            norm_x_w = '0;
            norm_y_w = ONE;
            norm_z_w = '0;
        end else begin
            norm_x_w = mul_nx_r;
            norm_y_w = mul_ny_r;
            norm_z_w = mul_nz_r;
        end
    end

    logic signed [WORD-1:0] prod0_r, prod1_r, prod2_r;
    logic signed [WORD-1:0] light_dist_r;

    // Reflection-math state: dot prod, doubled dot, r_term, refl.
    logic signed [WORD-1:0] p_dn_0_r, p_dn_1_r, p_dn_2_r;
    logic signed [WORD-1:0] dot_dn_r, two_dot_r;
    logic signed [WORD-1:0] r_term_x_r, r_term_y_r, r_term_z_r;
    logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;

    logic signed [WORD-1:0] mul_a, mul_b, mul_result;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));

    always_comb begin
        mul_a = '0; mul_b = '0;
        case (tick)
            6'd2: begin mul_a = diff_x_r; mul_b = inv_dm_r; end
            6'd3: begin mul_a = diff_y_r; mul_b = inv_dm_r; end
            6'd4: begin mul_a = diff_z_r; mul_b = inv_dm_r; end
```

```
            6'd5: begin mul_a = dx_r;     mul_b = mul_nx_r; end
            6'd6: begin mul_a = dy_r;     mul_b = mul_ny_r; end
            6'd7: begin mul_a = dz_r;     mul_b = mul_nz_r; end
            6'd8:  begin mul_a = tl_x_r;  mul_b = tl_x_r;   end
            6'd9:  begin mul_a = tl_y_r;  mul_b = tl_y_r;   end
            6'd10: begin mul_a = tl_z_r;  mul_b = tl_z_r;   end
            6'd11: begin mul_a = two_dot_r; mul_b = mul_nx_r; end
            6'd12: begin mul_a = two_dot_r; mul_b = mul_ny_r; end
            6'd13: begin mul_a = two_dot_r; mul_b = mul_nz_r; end
            default: begin end
        endcase
    end

    logic                  sqrt_start;
    logic signed [WORD-1:0] sqrt_in;
    logic signed [WORD-1:0] sqrt_out;
    logic                  sqrt_done;
    assign sqrt_start = (tick == 6'd12);
    assign sqrt_in    = prod0_r + prod1_r + prod2_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
        .a(sqrt_in), .result(sqrt_out), .done(sqrt_done)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            inv_dm_r <= '0;
            diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
            mul_nx_r <= '0; mul_ny_r <= '0; mul_nz_r <= '0;
            tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            dx_r <= '0; dy_r <= '0; dz_r <= '0;
            hit_type_r <= HIT_TYPE_SKY;
            bag_r <= '0;
            prod0_r <= '0; prod1_r <= '0; prod2_r <= '0;
            light_dist_r <= '0;
            p_dn_0_r <= '0; p_dn_1_r <= '0; p_dn_2_r <= '0;
            dot_dn_r <= '0; two_dot_r <= '0;
            r_term_x_r <= '0; r_term_y_r <= '0; r_term_z_r <= '0;
            refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
        end else begin
            case (tick)
                6'd0: begin
                    hit_x_r <= norm_hx; hit_y_r <= norm_hy; hit_z_r <= norm_hz;
                    inv_dm_r <= norm_inv_dm;
                    diff_x_r <= norm_dfx; diff_y_r <= norm_dfy; diff_z_r <= norm_dfz;
                    tl_x_r <= norm_tlx; tl_y_r <= norm_tly; tl_z_r <= norm_tlz;
                    col_r_r <= norm_cr; col_g_r <= norm_cg; col_b_r <= norm_cb;
                    dx_r <= norm_dx; dy_r <= norm_dy; dz_r <= norm_dz;
                    hit_type_r <= norm_hit_type;
                    bag_r <= norm_bag;
                end
                6'd3: mul_nx_r <= mul_result;
                6'd4: mul_ny_r <= mul_result;
                6'd5: mul_nz_r <= mul_result;
                6'd6: p_dn_0_r <= mul_result;
                6'd7: p_dn_1_r <= mul_result;
                6'd8: p_dn_2_r <= mul_result;
                6'd9: begin
                    automatic logic signed [WORD-1:0] dn_sum = p_dn_0_r + p_dn_1_r + p_dn_2_r;
                    dot_dn_r  <= dn_sum;
                    two_dot_r <= dn_sum <<< 1;
                    prod0_r   <= mul_result;
                end
                6'd10: prod1_r <= mul_result;
                6'd11: prod2_r <= mul_result;
                6'd12: r_term_x_r <= mul_result;
                6'd13: r_term_y_r <= mul_result;
                6'd14: begin
                    r_term_z_r <= mul_result;
                    refl_x_r   <= dx_r - r_term_x_r;
                end
                6'd15: refl_y_r <= dy_r - r_term_y_r;
```

```
                6'd16: refl_z_r <= dz_r - r_term_z_r;
                default: begin end
            endcase
            if (sqrt_done) light_dist_r <= sqrt_out;
        end
    end

    assign refl_hx = hit_x_r;
    assign refl_hy = hit_y_r;
    assign refl_hz = hit_z_r;
    assign refl_nx = norm_x_w;
    assign refl_ny = norm_y_w;
    assign refl_nz = norm_z_w;
    assign refl_tlx = tl_x_r;
    assign refl_tly = tl_y_r;
    assign refl_tlz = tl_z_r;
    assign refl_cr = col_r_r;
    assign refl_cg = col_g_r;
    assign refl_cb = col_b_r;
    assign refl_hit_type = hit_type_r;
    assign refl_ld = light_dist_r;
    assign refl_rx = refl_x_r;
    assign refl_ry = refl_y_r;
    assign refl_rz = refl_z_r;
    assign refl_bag = bag_r;
endmodule

// Stage F - LIGHT INV: emit 1/light_dist (single fp_inv) + ride-along forward.
// Stage G does the ldir = tl·inv_light_dist muls on its mul0 idle slots.
module rtl_light_dir_div
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                       // clock, async neg-edge reset
    input  logic [5:0]              tick,                              // pipe-local tick counter
    input  logic signed [WORD-1:0]  refl_hx, refl_hy, refl_hz,         // surface hit point (forwarded)
    input  logic signed [WORD-1:0]  refl_nx, refl_ny, refl_nz,         // unit normal (forwarded)
    input  logic signed [WORD-1:0]  refl_tlx, refl_tly, refl_tlz,      // to-light vector (forwarded; for ldir =
tl·inv_ld)
    input  logic signed [WORD-1:0]  refl_cr, refl_cg, refl_cb,         // base color (forwarded)
    input  logic [2:0]              refl_hit_type,                     // hit type (forwarded)
    input  logic signed [WORD-1:0]  refl_ld,                           // |to-light| from Stage E - divisor for
inv_ld
    input  logic signed [WORD-1:0]  refl_rx, refl_ry, refl_rz,         // reflection vector r (forwarded)
    input  pipe_baggage_t           refl_bag,                          // baggage
    output logic signed [WORD-1:0]  ldir_hx, ldir_hy, ldir_hz,         // hit point (forwarded)
    output logic signed [WORD-1:0]  ldir_nx, ldir_ny, ldir_nz,         // unit normal (forwarded)
    output logic signed [WORD-1:0]  ldir_tlx, ldir_tly, ldir_tlz,      // to-light vector (forwarded)
    output logic signed [WORD-1:0]  ldir_inv_ld,                       // 1/|to-light| - Stage G multiplies tl by
this (THIS STAGE'S WORK)
    output logic signed [WORD-1:0]  ldir_ld,                           // |to-light| (forwarded for shadow t-
comparison)
    output logic signed [WORD-1:0]  ldir_cr, ldir_cg, ldir_cb,         // base color (forwarded)
    output logic [2:0]              ldir_hit_type,                     // hit type (forwarded for shadow self-skip)
    output logic signed [WORD-1:0]  ldir_rx, ldir_ry, ldir_rz,         // reflection vector r (forwarded)
    output pipe_baggage_t           ldir_bag                           // baggage forwarded to Stage G
);
    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] nx_r, ny_r, nz_r;
    logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] light_dist_r;
    logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
    logic [2:0]             hit_type_r;
    pipe_baggage_t          bag_r;

    logic                   inv_start;
    assign inv_start = (tick == 6'd0);

    logic signed [WORD-1:0] inv_ld_w;
    logic                   inv_ld_done;
```

```
    fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_ld (
        .clk(clk), .rst_n(rst_n), .start(inv_start),
        .b(refl_ld), .result(inv_ld_w), .done(inv_ld_done)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            nx_r <= '0; ny_r <= '0; nz_r <= '0;
            tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            light_dist_r <= '0; hit_type_r <= HIT_TYPE_SKY;
            refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
            bag_r <= '0;
        end else if (tick == 6'd0) begin
            hit_x_r <= refl_hx; hit_y_r <= refl_hy; hit_z_r <= refl_hz;
            nx_r <= refl_nx; ny_r <= refl_ny; nz_r <= refl_nz;
            tl_x_r <= refl_tlx; tl_y_r <= refl_tly; tl_z_r <= refl_tlz;
            col_r_r <= refl_cr; col_g_r <= refl_cg; col_b_r <= refl_cb;
            light_dist_r <= refl_ld;
            refl_x_r <= refl_rx; refl_y_r <= refl_ry; refl_z_r <= refl_rz;
            hit_type_r <= refl_hit_type;
            bag_r <= refl_bag;
        end
    end

    assign ldir_hx = hit_x_r;
    assign ldir_hy = hit_y_r;
    assign ldir_hz = hit_z_r;
    assign ldir_nx = nx_r;
    assign ldir_ny = ny_r;
    assign ldir_nz = nz_r;
    assign ldir_tlx = tl_x_r;
    assign ldir_tly = tl_y_r;
    assign ldir_tlz = tl_z_r;
    assign ldir_inv_ld = inv_ld_w;
    assign ldir_ld = light_dist_r;
    assign ldir_cr = col_r_r;
    assign ldir_cg = col_g_r;
    assign ldir_cb = col_b_r;
    assign ldir_hit_type = hit_type_r;
    assign ldir_rx = refl_x_r;
    assign ldir_ry = refl_y_r;
    assign ldir_rz = refl_z_r;
    assign ldir_bag = bag_r;
endmodule

// Stage G - SHADOW TEST: shadow-ray quadratic + visibility check
module rtl_shadow_test
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int TAG_W     = 7
)(
    input  logic                    clk, rst_n,                      // clock, async neg-edge reset
    input  logic [5:0]              tick,                             // pipe-local tick counter
    input  logic signed [WORD-1:0]  ldir_hx, ldir_hy, ldir_hz,        // surface hit point (used as shadow ray
origin + bias)
    input  logic signed [WORD-1:0]  ldir_nx, ldir_ny, ldir_nz,        // unit normal (for shadow-origin bias hit +
 n/1024)
    input  logic signed [WORD-1:0]  ldir_tlx, ldir_tly, ldir_tlz,     // to-light vector (multiplied by inv_ld → 
ldir)
    input  logic signed [WORD-1:0]  ldir_inv_ld,                      // 1/|to-light| (from Stage F)
    input  logic signed [WORD-1:0]  ldir_ld,                          // |to-light| - compared with shadow ray's t
 to test occlusion
    input  logic signed [WORD-1:0]  ldir_cr, ldir_cg, ldir_cb,        // base color (forwarded)
    input  logic [2:0]              ldir_hit_type,                    // hit type (used for self-skip on shadow
ray)
    input  scene_t                  in_scene,                         // scene relay (reads .s0/.s1/.cone geometry
)
    input  logic signed [WORD-1:0]  ldir_rx, ldir_ry, ldir_rz,        // reflection vector r (forwarded)
    input  pipe_baggage_t           ldir_bag,                         // baggage
    output logic signed [WORD-1:0]  shdw_hx, shdw_hy, shdw_hz,        // hit point (forwarded)
```

```
    output logic signed [WORD-1:0]  shdw_nx, shdw_ny, shdw_nz,         // unit normal (forwarded)
    output logic signed [WORD-1:0]  shdw_lx, shdw_ly, shdw_lz,         // ldir = tl·inv_ld - unit light direction (computed here)
    output logic signed [WORD-1:0]  shdw_cr, shdw_cg, shdw_cb,         // base color (forwarded)
    output logic [2:0]              shdw_hit_type,                     // hit type (forwarded)
    output logic                    shdw_in_shadow,                    // 1 = light blocked by any primitive (THIS STAGE'S WORK)
    output logic signed [WORD-1:0]  shdw_rx, shdw_ry, shdw_rz,         // reflection vector r (forwarded to Stage H)
    output pipe_baggage_t           shdw_bag                           // baggage forwarded to Stage H
);
    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] nx_r, ny_r, nz_r;

    logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
    logic signed [WORD-1:0] inv_ld_r;
    logic signed [WORD-1:0] lx_r, ly_r, lz_r;
    logic signed [WORD-1:0] light_dist_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
    logic [2:0]             hit_type_r;
    pipe_baggage_t          bag_r;

    sphere_t sc0_local, sc1_local;
    cone_t   cone_local;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            sc0_local <= '0;
            sc1_local <= '0;
            cone_local <= '0;
        end else begin
            sc0_local <= in_scene.s0;
            sc1_local <= in_scene.s1;
            cone_local <= in_scene.cone;
        end
    end

    logic signed [WORD-1:0] sh_ox_c, sh_oy_c, sh_oz_c;
    logic signed [WORD-1:0] oc0_x_c, oc0_y_c, oc0_z_c;
    logic signed [WORD-1:0] oc1_x_c, oc1_y_c, oc1_z_c;
    logic signed [WORD-1:0] coc_x_c, coc_y_c, coc_z_c;
    assign sh_ox_c = hit_x_r + (nx_r >>> FRAC_BITS);
    assign sh_oy_c = hit_y_r + (ny_r >>> FRAC_BITS);
    assign sh_oz_c = hit_z_r + (nz_r >>> FRAC_BITS);
    assign oc0_x_c = sh_ox_c - sc0_local.x;
    assign oc0_y_c = sh_oy_c - sc0_local.y;
    assign oc0_z_c = sh_oz_c - sc0_local.z;
    assign oc1_x_c = sh_ox_c - sc1_local.x;
    assign oc1_y_c = sh_oy_c - sc1_local.y;
    assign oc1_z_c = sh_oz_c - sc1_local.z;
    assign coc_x_c = sh_ox_c - cone_local.apex_x;
    assign coc_y_c = sh_oy_c - cone_local.apex_y;
    assign coc_z_c = sh_oz_c - cone_local.apex_z;

    logic signed [WORD-1:0] oc0_x_r, oc0_y_r, oc0_z_r;
    logic signed [WORD-1:0] oc1_x_r, oc1_y_r, oc1_z_r;
    logic signed [WORD-1:0] coc_x_r, coc_y_r, coc_z_r;

    // Cone shadow state
    logic signed [WORD-1:0] cone_a_r, cone_b_r, cone_c_r;
    // Pre-registered negation of cone_b_r. cone_b_r is captured at tick=10
    // and stable thereafter, but `-cone_b_r - cone_disc_sqrt_r` (used by
    // the cone-num mux at tick=38 and num_w comb) was synthesized as two
    // 27-bit carry chains in series, eating ~3 ns. A free-running
    // neg_cone_b_r FF breaks one of the chains; the comb consumers see
    // `neg_cone_b_r - cone_disc_sqrt_r` instead.
    logic signed [WORD-1:0] neg_cone_b_r;
    logic signed [WORD-1:0] cone_dy_sq_r, cone_k_dy_sq_r;
    logic signed [WORD-1:0] cone_cdx_sq_r, cone_cdy_sq_r, cone_cdz_sq_r, cone_k_cdy_sq_r;
    logic signed [WORD-1:0] cone_cdx_dx_r, cone_cdy_dy_r, cone_cdz_dz_r, cone_k_cdy_dy_r;
    logic signed [WORD-1:0] cone_b_sq_r, cone_a_c_r;
    logic signed [WORD-1:0] cone_disc_sqrt_r;
    logic                   cone_no_disc_r;        // disc < 0 → no shadow hit

```

```
    logic signed [WORD-1:0] cone_a_ld_r;          // a · light_dist
    logic signed [WORD-1:0] cone_a_sh_oy_r;       // a · sh_origin.y
    logic signed [WORD-1:0] cone_a_slab_bot_r;    // a · (apex_y - height)
    logic signed [WORD-1:0] cone_a_slab_top_r;    // a · (apex_y - CONE_APEX_CULL)
    logic signed [WORD-1:0] cone_num_ly_r;        // num · ldir.y, where num = -b - sqrt_disc
    logic                   cone_in_shadow_r;

    // Two fp_muls
    logic signed [WORD-1:0] m0_a, m0_b, m0_r;
    logic signed [WORD-1:0] m1_a, m1_b, m1_r;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul0 (.clk(clk), .a(m0_a), .b(m0_b), .result(m0_r));
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul1 (.clk(clk), .a(m1_a), .b(m1_b), .result(m1_r));

    // Third fp_mul dedicated to cone shadow quadratic
    logic signed [WORD-1:0] mc_a, mc_b, mc_r;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone_shdw (.clk(clk), .a(mc_a), .b(mc_b), .result(mc_r));

    // Free-running neg_cone_b_r relay: track -cone_b_r with 1-cycle lag.
    // cone_b_r settles at tick=11 and is stable for the rest of the ray,
    // so neg_cone_b_r is valid from tick=12 onward - well before the
    // tick=38 cone-num mux that consumes it.
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) neg_cone_b_r <= '0;
        else        neg_cone_b_r <= -cone_b_r;
    end

    // Per-DSP 1-cycle-delayed tick replicas
    (* preserve, dont_merge *) logic [5:0] tick_m0_r;
    (* preserve, dont_merge *) logic [5:0] tick_m1_r;
    (* preserve, dont_merge *) logic [5:0] tick_mc_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            tick_m0_r <= '0;
            tick_m1_r <= '0;
            tick_mc_r <= '0;
        end else begin
            tick_m0_r <= tick;
            tick_m1_r <= tick;
            tick_mc_r <= tick;
        end
    end

    // No fp_inv needed - shadow check uses a-scaled cross-multiplication
    // (cone_a_r · light_dist_r, etc.) so the divide is avoided. fp_inv's
    // 39c latency would land past the in_shadow_r capture deadline.

    // fp_sqrt on cone discriminant. Kicked at t=18 (b_sq_r valid from t=17,
    // a_c_r valid from t=18, disc combinational at t=18 - t=18 is the earliest
    // possible start without unwinding the cone_k_dy_sq_r race fix in 649c146).
    // 21c latency (cocotb-verified) → done at tick=39 → cone_disc_sqrt_r register
    // valid from t=40. The num·ly mul and cone_in_shadow_r capture sit at t=40
    // and t=42 respectively so they consume the fresh disc.
    logic                   sqrt_cone_start;
    logic signed [WORD-1:0] sqrt_cone_in;
    logic signed [WORD-1:0] sqrt_cone_out;
    logic                   sqrt_cone_done;
    assign sqrt_cone_start = (tick == 6'd18);
    assign sqrt_cone_in    = cone_b_sq_r - cone_a_c_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt_cone_shdw (
        .clk(clk), .rst_n(rst_n), .start(sqrt_cone_start),
        .a(sqrt_cone_in), .result(sqrt_cone_out), .done(sqrt_cone_done)
    );
    logic signed [WORD-1:0] p_b0_0_r, p_b1_0_r, p_b2_0_r;
    logic signed [WORD-1:0] p_c0_0_r, p_c1_0_r, p_c2_0_r;
    logic signed [WORD-1:0] half_b0_r, c0_r;
    logic signed [WORD-1:0] half_b0_sq_r;
    logic signed [WORD-1:0] disc_sqrt0_r;
    logic                   no_hit0_raw_r;

    logic signed [WORD-1:0] p_b0_1_r, p_b1_1_r, p_b2_1_r;
    logic signed [WORD-1:0] p_c0_1_r, p_c1_1_r, p_c2_1_r;
    logic signed [WORD-1:0] half_b1_r, c1_r;
    logic signed [WORD-1:0] half_b1_sq_r;
    logic signed [WORD-1:0] disc_sqrt1_r;
```

```
    logic                  no_hit1_raw_r;

    // mul0 t0..2 use upstream ldir_tlx/ldir_inv_ld at t=0 (locals not latched yet)
    // and locals at t=1/2 (upstream regs flipped to next ray). Same pattern
    // as stage E's diff·inv_dm.
    // m0 (sphere 0 shadow) operand mux - driven by tick_m0_r. Case arms
    // shift -1 vs the original tick-driven cases; t=0 wraps to II-1=42.
    always_comb begin
        m0_a = '0; m0_b = '0;
        case (tick_m0_r)
            6'd42: begin m0_a = ldir_tlx;  m0_b = ldir_inv_ld; end  // orig t=0
            6'd0:  begin m0_a = tl_y_r;    m0_b = inv_ld_r;    end  // orig t=1
            6'd1:  begin m0_a = tl_z_r;    m0_b = inv_ld_r;    end  // orig t=2
            6'd4:  begin m0_a = lx_r;      m0_b = oc0_x_r;     end  // orig t=5
            6'd5:  begin m0_a = ly_r;      m0_b = oc0_y_r;     end  // orig t=6
            6'd6:  begin m0_a = lz_r;      m0_b = oc0_z_r;     end  // orig t=7
            6'd7:  begin m0_a = oc0_x_r;   m0_b = oc0_x_r;     end  // orig t=8
            6'd8:  begin m0_a = oc0_y_r;   m0_b = oc0_y_r;     end  // orig t=9
            6'd9:  begin m0_a = oc0_z_r;   m0_b = oc0_z_r;     end  // orig t=10
            6'd10: begin m0_a = half_b0_r; m0_b = half_b0_r;   end  // orig t=11
            default: begin end
        endcase
    end

    always_comb begin
        m1_a = '0; m1_b = '0;
        case (tick_m1_r)
            6'd1:  begin m1_a = lx_r;      m1_b = oc1_x_r;   end  // orig t=2
            6'd2:  begin m1_a = ly_r;      m1_b = oc1_y_r;   end  // orig t=3
            6'd3:  begin m1_a = lz_r;      m1_b = oc1_z_r;   end  // orig t=4
            6'd4:  begin m1_a = oc1_x_r;   m1_b = oc1_x_r;   end  // orig t=5
            6'd5:  begin m1_a = oc1_y_r;   m1_b = oc1_y_r;   end  // orig t=6
            6'd6:  begin m1_a = oc1_z_r;   m1_b = oc1_z_r;   end  // orig t=7
            6'd7:  begin m1_a = half_b1_r; m1_b = half_b1_r; end  // orig t=8
            default: begin end
        endcase
    end

    // mc (cone shadow) operand mux - driven by tick_mc_r
    always_comb begin
        mc_a = '0; mc_b = '0;
        case (tick_mc_r)
            6'd2:  begin                                          // orig t=3
                mc_a = ly_r;      mc_b = ly_r;                    // ldir.y² → cap t=4
            end
            6'd4:  begin                                          // orig t=5
                mc_a = coc_x_r;   mc_b = coc_x_r;                 // Δ.x² → cap t=6
            end
            6'd5:  begin                                          // orig t=6
                mc_a = coc_y_r;   mc_b = coc_y_r;                 // Δ.y² → cap t=7
            end
            6'd6:  begin                                          // orig t=7
                mc_a = coc_z_r;   mc_b = coc_z_r;                 // Δ.z² → cap t=8
            end
            6'd7:  begin                                          // orig t=8
                mc_a = cone_local.k_sq;
                mc_b = cone_cdy_sq_r;                             // Δk²·.y² → cap t=9
            end
            6'd8:  begin                                          // orig t=9
                mc_a = coc_x_r;   mc_b = lx_r;                    // Δ.x·D.x → cap t=10
            end
            6'd9:  begin                                          // orig t=10
                mc_a = coc_y_r;   mc_b = ly_r;                    // Δ.y·D.y → cap t=11
            end
            6'd10: begin                                          // orig t=11
                mc_a = coc_z_r;   mc_b = lz_r;                    // Δ.z·D.z → cap t=12
            end
            6'd11: begin                                          // orig t=12
                mc_a = cone_local.k_sq;
                mc_b = cone_cdy_dy_r;                             // k²·Δ(.y·D.y) → cap t=13
            end
            6'd12: begin                                          // orig t=13
                mc_a = cone_local.k_sq;
                mc_b = cone_dy_sq_r;                              // k²·D.y² → cap t=14
```

```
            end
            6'd14: begin                                            // orig t=15
                mc_a = cone_b_w_r; mc_b = cone_b_w_r;               // b² → cap t=16
            end
            6'd15: begin                                            // orig t=16
                mc_a = cone_a_r; mc_b = cone_c_r;                   // a·c → cap t=17
            end
            6'd16: begin                                            // orig t=17
                mc_a = cone_a_r; mc_b = light_dist_r;               // a · ld → cap t=18
            end
            6'd18: begin                                            // orig t=19
                mc_a = cone_a_r; mc_b = sh_oy_c;                    // a · sh_oy → cap t=20
            end
            6'd19: begin                                            // orig t=20
                mc_a = cone_a_r;
                mc_b = apex_y_minus_height_r;                       // a · slab_bot → cap t=21
            end
            6'd20: begin                                            // orig t=21
                mc_a = cone_a_r;
                mc_b = apex_y_minus_cull_r;                         // a · slab_top → cap t=22
            end
            6'd39: begin                                            // orig t=40
                // Pushed +2 ticks vs the prior schedule: fp_sqrt is 21c (cocotb
                // verified), so sqrt_cone_done fires at t=39 and cone_disc_sqrt_r
                // only becomes valid during t=40. Reading it at t=38 (the old
                // slot) captured the previous ray's disc - visible as thin arcs
                // at cone-shadow boundaries.
                mc_a = neg_cone_b_r - cone_disc_sqrt_r;            // -cone_b pre-registered (saves a carry chain)
                mc_b = ly_r;                                        // num · ldir.y → cap t=41
            end
            default: begin end
        endcase
    end

    localparam signed [WORD-1:0] CONE_APEX_CULL_SHDW = 1 <<< (FRAC_BITS - 5);

    logic signed [WORD-1:0] apex_y_minus_height_r, apex_y_minus_cull_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            apex_y_minus_height_r <= '0;
            apex_y_minus_cull_r   <= '0;
        end else begin
            apex_y_minus_height_r <= cone_local.apex_y - cone_local.height;
            apex_y_minus_cull_r   <= cone_local.apex_y - CONE_APEX_CULL_SHDW;
        end
    end

    logic signed [WORD-1:0] cone_a_w, cone_b_w, cone_c_w;
    localparam signed [WORD-1:0] ONE_FX_LOCAL = 1 <<< FRAC_BITS;
    assign cone_a_w = ONE_FX_LOCAL - cone_dy_sq_r - cone_k_dy_sq_r;
    assign cone_b_w = cone_cdx_dx_r + cone_cdz_dz_r - cone_k_cdy_dy_r;
    assign cone_c_w = cone_cdx_sq_r + cone_cdz_sq_r - cone_k_cdy_sq_r;

    logic signed [WORD-1:0] cone_b_w_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) cone_b_w_r <= '0;
        else        cone_b_w_r <= cone_b_w;
    end

    // sqrt_start at t=15. fp_sqrt is 21c →startdone (cocotb-verified), so done
    // fires at tick=36 and disc_sqrt0/1_r capture at edge-end-36 (= valid from
    // t=37). The t=37 in_shadow_r capture then reads the new disc in the same
    // cycle. Was t=16 before, which made the disc-update and in_shadow_r-capture
    // share an edge - in_shadow_r captured the PREVIOUS ray's disc, causing
    // thin shadow-boundary arcs in reflections.
    logic                  sqrt_start;
    logic signed [WORD-1:0] sqrt0_in, sqrt1_in;
    logic signed [WORD-1:0] sqrt0_out, sqrt1_out;
    logic                  sqrt0_done, sqrt1_done;
    assign sqrt_start = (tick == 6'd15);
    assign sqrt0_in   = half_b0_sq_r - c0_r;
    assign sqrt1_in   = half_b1_sq_r - c1_r;
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt0 (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
```

```
        .a(sqrt0_in), .result(sqrt0_out), .done(sqrt0_done)
    );
    fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt1 (
        .clk(clk), .rst_n(rst_n), .start(sqrt_start),
        .a(sqrt1_in), .result(sqrt1_out), .done(sqrt1_done)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            nx_r <= '0; ny_r <= '0; nz_r <= '0;
            tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
            inv_ld_r <= '0;
            lx_r <= '0; ly_r <= '0; lz_r <= '0;
            light_dist_r <= '0;
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
            hit_type_r <= HIT_TYPE_SKY;
            bag_r <= '0;
            oc0_x_r <= '0; oc0_y_r <= '0; oc0_z_r <= '0;
            oc1_x_r <= '0; oc1_y_r <= '0; oc1_z_r <= '0;
            coc_x_r <= '0; coc_y_r <= '0; coc_z_r <= '0;
            p_b0_0_r <= '0; p_b1_0_r <= '0; p_b2_0_r <= '0;
            p_c0_0_r <= '0; p_c1_0_r <= '0; p_c2_0_r <= '0;
            half_b0_r <= '0; c0_r <= '0;
            half_b0_sq_r <= '0;
            disc_sqrt0_r <= '0; no_hit0_raw_r <= 1'b0;
            p_b0_1_r <= '0; p_b1_1_r <= '0; p_b2_1_r <= '0;
            p_c0_1_r <= '0; p_c1_1_r <= '0; p_c2_1_r <= '0;
            half_b1_r <= '0; c1_r <= '0;
            half_b1_sq_r <= '0;
            disc_sqrt1_r <= '0; no_hit1_raw_r <= 1'b0;
            cone_a_r <= '0; cone_b_r <= '0; cone_c_r <= '0;
            cone_dy_sq_r <= '0; cone_k_dy_sq_r <= '0;
            cone_cdx_sq_r <= '0; cone_cdy_sq_r <= '0; cone_cdz_sq_r <= '0;
            cone_k_cdy_sq_r <= '0;
            cone_cdx_dx_r <= '0; cone_cdy_dy_r <= '0; cone_cdz_dz_r <= '0;
            cone_k_cdy_dy_r <= '0;
            cone_b_sq_r <= '0; cone_a_c_r <= '0;
            cone_disc_sqrt_r <= '0; cone_no_disc_r <= 1'b0;
            cone_a_ld_r       <= '0;
            cone_a_sh_oy_r    <= '0;
            cone_a_slab_bot_r <= '0;
            cone_a_slab_top_r <= '0;
            cone_num_ly_r     <= '0;
            cone_in_shadow_r  <= 1'b0;
            in_shadow_r  <= 1'b0;
        end else begin
            case (tick)
                6'd0: begin
                    hit_x_r <= ldir_hx; hit_y_r <= ldir_hy; hit_z_r <= ldir_hz;
                    nx_r <= ldir_nx; ny_r <= ldir_ny; nz_r <= ldir_nz;
                    tl_x_r <= ldir_tlx; tl_y_r <= ldir_tly; tl_z_r <= ldir_tlz;
                    inv_ld_r <= ldir_inv_ld;
                    light_dist_r <= ldir_ld;
                    col_r_r <= ldir_cr; col_g_r <= ldir_cg; col_b_r <= ldir_cb;
                    refl_x_r <= ldir_rx; refl_y_r <= ldir_ry; refl_z_r <= ldir_rz;
                    hit_type_r <= ldir_hit_type;
                    bag_r <= ldir_bag;
                end
                6'd1: begin
                    oc0_x_r <= oc0_x_c; oc0_y_r <= oc0_y_c; oc0_z_r <= oc0_z_c;
                    oc1_x_r <= oc1_x_c; oc1_y_r <= oc1_y_c; oc1_z_r <= oc1_z_c;
                    coc_x_r <= coc_x_c; coc_y_r <= coc_y_c; coc_z_r <= coc_z_c;
                    lx_r <= m0_r;
                end
                6'd2: ly_r <= m0_r;
                6'd3: begin
                    p_b0_1_r <= m1_r;
                    lz_r <= m0_r;
                end
                6'd4: begin p_b1_1_r <= m1_r; cone_dy_sq_r    <= mc_r; end
                6'd5: begin p_b2_1_r <= m1_r; end
                // cone_k_dy_sq_r capture moved to t=14 (race fix - see mux comment)
```

```
                6'd6: begin
                    p_b0_0_r <= m0_r; p_c0_1_r <= m1_r;
                    half_b1_r <= p_b0_1_r + p_b1_1_r + p_b2_1_r;
                    cone_cdx_sq_r <= mc_r;
                end
                6'd7: begin p_b1_0_r <= m0_r; p_c1_1_r <= m1_r; cone_cdy_sq_r <= mc_r; end
                6'd8: begin p_b2_0_r <= m0_r; p_c2_1_r <= m1_r; cone_cdz_sq_r <= mc_r; end
                6'd9: begin
                    p_c0_0_r <= m0_r; half_b1_sq_r <= m1_r;
                    half_b0_r <= p_b0_0_r + p_b1_0_r + p_b2_0_r;
                    c1_r      <= p_c0_1_r + p_c1_1_r + p_c2_1_r - sc1_local.r_sq;
                    cone_k_cdy_sq_r <= mc_r;
                end
                6'd10: begin p_c1_0_r <= m0_r; cone_cdx_dx_r <= mc_r; end
                6'd11: begin p_c2_0_r <= m0_r; cone_cdy_dy_r <= mc_r; end
                6'd12: begin
                    half_b0_sq_r <= m0_r;
                    c0_r         <= p_c0_0_r + p_c1_0_r + p_c2_0_r - sc0_local.r_sq;
                    no_hit1_raw_r <= (half_b1_sq_r < c1_r);
                    cone_cdz_dz_r <= mc_r;
                end
                6'd13: cone_k_cdy_dy_r <= mc_r;
                6'd14: begin
                    cone_b_r          <= cone_b_w;
                    cone_c_r          <= cone_c_w;
                    cone_k_dy_sq_r    <= mc_r;             // race fix capture (was t=5)
                end
                6'd15: begin
                    no_hit0_raw_r <= (half_b0_sq_r < c0_r);
                    cone_a_r      <= cone_a_w;             // pushed +1 - needs settled cone_k_dy_sq_r
                end
                6'd16: cone_b_sq_r <= mc_r;                // b² kick at tick_mc_r=14

                6'd17: begin
                    cone_a_c_r     <= mc_r;
                    cone_no_disc_r <= (cone_b_sq_r < mc_r);
                end
                6'd18: cone_a_ld_r       <= mc_r;
                6'd20: cone_a_sh_oy_r    <= mc_r;
                6'd21: cone_a_slab_bot_r <= mc_r;
                6'd22: cone_a_slab_top_r <= mc_r;
                6'd37: in_shadow_r <= in_shadow_w;
                6'd41: cone_num_ly_r <= mc_r;     // num · ldir.y (mc op moved to t=40)
                6'd42: cone_in_shadow_r <= cone_blocks_w;
                default: begin end
            endcase
            if (sqrt0_done) disc_sqrt0_r <= sqrt0_out;
            if (sqrt1_done) disc_sqrt1_r <= sqrt1_out;
            if (sqrt_cone_done) cone_disc_sqrt_r <= sqrt_cone_out;
        end
    end

    // Shadow visibility check
    logic no_hit0_w, no_hit1_w;
    assign no_hit0_w = no_hit0_raw_r || (hit_type_r == HIT_TYPE_S0);
    assign no_hit1_w = no_hit1_raw_r || (hit_type_r == HIT_TYPE_S1);

    logic signed [WORD-1:0] t0_s0_w, t0_s1_w;
    assign t0_s0_w = -half_b0_r - disc_sqrt0_r;
    assign t0_s1_w = -half_b1_r - disc_sqrt1_r;

    logic s0_blocks_w, s1_blocks_w;
    assign s0_blocks_w = !no_hit0_w && (t0_s0_w > 0) && (t0_s0_w < light_dist_r);
    assign s1_blocks_w = !no_hit1_w && (t0_s1_w > 0) && (t0_s1_w < light_dist_r);

    logic in_shadow_w;
    assign in_shadow_w = (hit_type_r != HIT_TYPE_SKY) && (s0_blocks_w || s1_blocks_w);

    logic in_shadow_r;

    logic signed [WORD-1:0] num_w;
    logic signed [WORD-1:0] cone_a_hit_y_w;
    assign num_w           = neg_cone_b_r - cone_disc_sqrt_r;
    assign cone_a_hit_y_w  = cone_a_sh_oy_r + cone_num_ly_r;
```

```
2908
2909        logic cone_a_pos, cone_a_neg;
2910        assign cone_a_pos = (cone_a_r > 0);
2911        assign cone_a_neg = (cone_a_r < 0);
2912
2913        logic cone_has_t_w;
2914        assign cone_has_t_w =
2915            (cone_a_pos && (num_w > 0) && (num_w < cone_a_ld_r)) ||
2916            (cone_a_neg && (num_w < 0) && (num_w > cone_a_ld_r));
2917
2918        logic cone_in_slab_w;
2919        assign cone_in_slab_w =
2920            (cone_a_pos && (cone_a_hit_y_w >= cone_a_slab_bot_r) &&
2921                           (cone_a_hit_y_w <= cone_a_slab_top_r)) ||
2922            (cone_a_neg && (cone_a_hit_y_w >= cone_a_slab_top_r) &&
2923                           (cone_a_hit_y_w <= cone_a_slab_bot_r));
2924
2925        logic cone_blocks_w;
2926        assign cone_blocks_w = (hit_type_r != HIT_TYPE_SKY) &&
2927                               (hit_type_r != HIT_TYPE_CONE) &&     // self-skip
2928                               !cone_no_disc_r &&                   // disc  0
2929                               cone_has_t_w &&
2930                               cone_in_slab_w;
2931
2932        assign shdw_hx = hit_x_r;
2933        assign shdw_hy = hit_y_r;
2934        assign shdw_hz = hit_z_r;
2935        assign shdw_nx = nx_r;
2936        assign shdw_ny = ny_r;
2937        assign shdw_nz = nz_r;
2938        assign shdw_lx = lx_r;
2939        assign shdw_ly = ly_r;
2940        assign shdw_lz = lz_r;
2941        assign shdw_cr = col_r_r;
2942        assign shdw_cg = col_g_r;
2943        assign shdw_cb = col_b_r;
2944        assign shdw_hit_type = hit_type_r;
2945        assign shdw_in_shadow = in_shadow_r || cone_in_shadow_r;
2946        assign shdw_rx = refl_x_r;
2947        assign shdw_ry = refl_y_r;
2948        assign shdw_rz = refl_z_r;
2949        assign shdw_bag = bag_r;
2950    endmodule
2951
2952    // Stage H - SHADE: hit_type/in_shadow mux selects sky / ambient / lit.
2953    //   sky:    color = passed-through pre-baked gradient
2954    //   shadow: color = clamp01(col·AMBIENT)
2955    //   else:   color = clamp01(col·AMBIENT + col·diff_coeff),
2956    //           diff_coeff = max(0, dot(n, ldir))
2957    module rtl_shade_blend
2958        import raytracer_pkg::*;
2959    #(
2960        parameter int FRAC_BITS = 13,
2961        parameter int WORD      = 27,
2962        parameter int TAG_W     = 7
2963    )(
2964        input  logic                    clk, rst_n,                      // clock, async neg-edge reset
2965        input  logic [5:0]              tick,                             // pipe-local tick counter
2966        input  logic signed [WORD-1:0]  shdw_hx, shdw_hy, shdw_hz,        // hit point (forwarded for spawn origin)
2967        input  logic signed [WORD-1:0]  shdw_nx, shdw_ny, shdw_nz,        // unit normal (used for Lambert dot)
2968        input  logic signed [WORD-1:0]  shdw_lx, shdw_ly, shdw_lz,        // unit light direction (used for Lambert
    dot)
2969        input  logic signed [WORD-1:0]  shdw_cr, shdw_cg, shdw_cb,        // base surface RGB
2970        input  logic [2:0]              shdw_hit_type,                    // hit type (sky/sphere/plane/cone - gates
    which branch)
2971        input  logic                    shdw_in_shadow,                   // 1 = use ambient only; 0 = ambient +
    diffuse
2972        input  logic signed [WORD-1:0]  shdw_rx, shdw_ry, shdw_rz,        // reflection vector r (forwarded for spawn
    direction)
2973        input  pipe_baggage_t           shdw_bag,                         // baggage
2974        output logic signed [WORD-1:0]  shade_cr, shade_cg, shade_cb,     // final shaded RGB (clamped) - THIS STAGE'S
     WORK
2975        output logic signed [WORD-1:0]  shade_hx, shade_hy, shade_hz,     // hit point (forwarded for spawn origin)
```

```
    output logic signed [WORD-1:0]  shade_rx, shade_ry, shade_rz,      // reflection direction (forwarded for spawn child ray)
    output logic [2:0]              shade_hit_type,                    // hit type (forwarded for spawn-eligibility check)
    output pipe_baggage_t           shade_bag                          // baggage (drop_out latched at tick=15 here)
);
    localparam signed [WORD-1:0] ONE     = 1 <<< FRAC_BITS;
    // AMBIENT 5/32 = (col>>3) + (col>>5). One extra adder vs 1/8, kept
    // per user pref for the less-crushed shadow tone.
    localparam signed [WORD-1:0] AMBIENT = (5 * ONE) / 32;

    logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
    logic signed [WORD-1:0] nx_r, ny_r, nz_r;
    logic signed [WORD-1:0] lx_r, ly_r, lz_r;
    logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
    logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
    logic [2:0]             hit_type_r;
    logic                   in_shadow_r;
    pipe_baggage_t          bag_r;

    logic signed [WORD-1:0] p0_r, p1_r, p2_r;
    logic signed [WORD-1:0] diff_coeff_r;
    logic signed [WORD-1:0] amb_r_r, amb_g_r, amb_b_r;
    logic signed [WORD-1:0] dif_r_r, dif_g_r, dif_b_r;
    // pre_clamp_*_r holds branch-combined value before clamp01. Splitting
    // combine and clamp into t=13/14 breaks a long →→tickaddcompare path.
    logic signed [WORD-1:0] pre_clamp_r_r, pre_clamp_g_r, pre_clamp_b_r;
    logic signed [WORD-1:0] color_r_rg, color_g_rg, color_b_rg;

    logic signed [WORD-1:0] mul_a, mul_b, mul_result;
    fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));

    logic [5:0] tick_local;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) tick_local <= 6'd0;
        else        tick_local <= tick;
    end

    always_comb begin
        mul_a = '0; mul_b = '0;
        case (tick_local)
            6'd1:  begin mul_a = nx_r;    mul_b = lx_r;         end
            6'd2:  begin mul_a = ny_r;    mul_b = ly_r;         end
            6'd3:  begin mul_a = nz_r;    mul_b = lz_r;         end
            6'd8:  begin mul_a = col_r_r; mul_b = diff_coeff_r; end
            6'd9:  begin mul_a = col_g_r; mul_b = diff_coeff_r; end
            6'd10: begin mul_a = col_b_r; mul_b = diff_coeff_r; end
            default: begin end
        endcase
    end

    logic signed [2*WORD-1:0] amb_r_full_w, amb_g_full_w, amb_b_full_w;
    assign amb_r_full_w = col_r_r * AMBIENT;
    assign amb_g_full_w = col_g_r * AMBIENT;
    assign amb_b_full_w = col_b_r * AMBIENT;
    logic signed [WORD-1:0] amb_r_w, amb_g_w, amb_b_w;
    assign amb_r_w = amb_r_full_w[WORD-1+FRAC_BITS:FRAC_BITS];
    assign amb_g_w = amb_g_full_w[WORD-1+FRAC_BITS:FRAC_BITS];
    assign amb_b_w = amb_b_full_w[WORD-1+FRAC_BITS:FRAC_BITS];

    function automatic logic signed [WORD-1:0] clamp01(input logic signed [WORD-1:0] v);
        if (v < 0)        return '0;
        else if (v > ONE) return ONE;
        else              return v;
    endfunction

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
            nx_r <= '0; ny_r <= '0; nz_r <= '0;
            lx_r <= '0; ly_r <= '0; lz_r <= '0;
            col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
            refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
```

```
            hit_type_r <= HIT_TYPE_SKY; in_shadow_r <= 1'b0;
            bag_r <= '0;
            p0_r <= '0; p1_r <= '0; p2_r <= '0;
            diff_coeff_r <= '0;
            amb_r_r <= '0; amb_g_r <= '0; amb_b_r <= '0;
            dif_r_r <= '0; dif_g_r <= '0; dif_b_r <= '0;
            pre_clamp_r_r <= '0; pre_clamp_g_r <= '0; pre_clamp_b_r <= '0;
            color_r_rg <= '0; color_g_rg <= '0; color_b_rg <= '0;
        end else begin
            case (tick)
                6'd0: begin
                    hit_x_r <= shdw_hx; hit_y_r <= shdw_hy; hit_z_r <= shdw_hz;
                    nx_r <= shdw_nx; ny_r <= shdw_ny; nz_r <= shdw_nz;
                    lx_r <= shdw_lx; ly_r <= shdw_ly; lz_r <= shdw_lz;
                    col_r_r <= shdw_cr; col_g_r <= shdw_cg; col_b_r <= shdw_cb;
                    refl_x_r <= shdw_rx; refl_y_r <= shdw_ry; refl_z_r <= shdw_rz;
                    hit_type_r <= shdw_hit_type;
                    in_shadow_r <= shdw_in_shadow;
                    bag_r <= shdw_bag;
                end
                6'd3: p0_r <= mul_result;
                6'd4: p1_r <= mul_result;
                6'd5: p2_r <= mul_result;
                6'd6: begin
                    automatic logic signed [WORD-1:0] dot_sum = p0_r + p1_r + p2_r;
                    diff_coeff_r <= dot_sum[WORD-1] ? '0 : dot_sum;
                end
                6'd1: begin
                    amb_r_r <= amb_r_w;
                    amb_g_r <= amb_g_w;
                    amb_b_r <= amb_b_w;
                end
                6'd10: dif_r_r <= mul_result;
                6'd11: dif_g_r <= mul_result;
                6'd12: dif_b_r <= mul_result;
                6'd13: begin
                    if (hit_type_r == HIT_TYPE_SKY) begin
                        pre_clamp_r_r <= col_r_r;
                        pre_clamp_g_r <= col_g_r;
                        pre_clamp_b_r <= col_b_r;
                    end else if (in_shadow_r) begin
                        pre_clamp_r_r <= amb_r_r;
                        pre_clamp_g_r <= amb_g_r;
                        pre_clamp_b_r <= amb_b_r;
                    end else begin
                        pre_clamp_r_r <= amb_r_r + dif_r_r;
                        pre_clamp_g_r <= amb_g_r + dif_g_r;
                        pre_clamp_b_r <= amb_b_r + dif_b_r;
                    end
                end
                6'd14: begin
                    color_r_rg <= clamp01(pre_clamp_r_r);
                    color_g_rg <= clamp01(pre_clamp_g_r);
                    color_b_rg <= clamp01(pre_clamp_b_r);
                end
                default: begin end
            endcase
        end
    end

    assign shade_cr = color_r_rg;
    assign shade_cg = color_g_rg;
    assign shade_cb = color_b_rg;
    assign shade_hx = hit_x_r;
    assign shade_hy = hit_y_r;
    assign shade_hz = hit_z_r;
    assign shade_rx = refl_x_r;
    assign shade_ry = refl_y_r;
    assign shade_rz = refl_z_r;
    assign shade_hit_type = hit_type_r;
    assign shade_bag = bag_r;
endmodule

// ============================================================================
```

```
// rtl_pipe - one fixed-II=43 raytracer pipeline. Chains stages A..H with a
// shared `tick` counter (wraps 0..II-1) and a per-pipe load_in handshake.
// PHASE parameter offsets the counter at reset so K parallel rtl_pipe
// instances can be staggered (PHASE = II·i/K). Output bus (drop_out,
// out_tag, out_color_*, out_refl_*, out_meta) is registered at t=15 from
// stage H's settled color + spawn side-band.
// ============================================================================
module rtl_pipe
    import raytracer_pkg::*;
#(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int WIDTH     = 480,
    parameter int HEIGHT    = 360,
    parameter int TAG_W     = 7,
    parameter int II        = 43,
    parameter int PHASE     = 0       // initial tick offset (for K-staggering)
)(
    input  logic                      clk, rst_n,                                // clock, async neg-edge reset
    input  logic                      load_in,                                    // 1c pulse to issue a new ray
at t=0 (=sv bit)
    input  logic signed [WORD-1:0]    in_light_x, in_light_y, in_light_z,         // light position (batch-stable)
    input  scene_t                    in_scene,                                   // per-pipe scene relay (
geometry + colors + reflect + floor)
    input  logic [2:0]                max_depth_r,                                // reflection recursion cap (0
disables)
    input  logic signed [WORD-1:0]    in_K_const,                                 // |light|²·63/64 - sun
threshold
    input  logic [TAG_W-1:0]          in_tag,                                     // lane tag for accumulator BRAM
    input  ray_meta_t                 in_meta,                                    // ray meta: depth, excluded,
total_shift
    // Reflection override: in_use_override=1 supplies parent hit + refl dir,
    // bypassing stage A's pixel/basis path.
    input  logic                      in_use_override,                            // 1 = spawn (use in_dir +
in_orig); 0 = primary
    input  logic signed [WORD-1:0]    in_orig_x, in_orig_y, in_orig_z,            // ray origin (camera or parent
hit point)
    input  logic signed [WORD-1:0]    in_dir_x, in_dir_y, in_dir_z,               // override direction (parent
reflection)
    // Shared basis_compute output from rtl_batch_pipe (single instance per
    // batch - replaces K-fold redundant per-pipe basis math).
    input  logic signed [WORD-1:0]    in_basis_raw_x, in_basis_raw_y, in_basis_raw_z, // primary ray direction (
right·px + up·py + fwd)
    output logic                      drop_out,                                   // 1 when stage H emits a real
ray (1-cycle pulse at tick=15)
    output logic [TAG_W-1:0]          out_tag,                                    // lane tag at the drop_out edge
    output logic signed [WORD-1:0]    out_color_r, out_color_g, out_color_b,      // shaded color at the drop_out
edge
    output ray_meta_t                 out_meta,                                   // ray meta at the drop_out edge

    output logic signed [WORD-1:0]    out_hit_x, out_hit_y, out_hit_z,            // parent hit point - child's
origin if spawning
    output logic signed [WORD-1:0]    out_refl_x, out_refl_y, out_refl_z,         // reflection direction r -
child's direction if spawning
    output logic [2:0]                out_hit_type,                               // hit type - selects which
reflect_shift to use

    output logic                      out_spawn_push,                             // drop_out & spawn_decision (
push into pipe's spawn_slot)
    output ray_meta_t                 out_spawn_meta,                             // child's depth+1, excluded,
total_shift+reflect_shift
    output logic                      at_load_tick                                // 1 during the cycle tick==0 (
arbiter signal)
);
    // Tick counter (wraps at II-1 -> 0)
    logic [5:0] tick;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            tick <= PHASE[5:0];
        else if (tick == II[5:0] - 1)
            tick <= 6'd0;
        else
            tick <= tick + 6'd1;
    end
```

```
3178
3179    logic signed [WORD-1:0] light_x_local, light_y_local, light_z_local;
3180    always_ff @(posedge clk or negedge rst_n) begin
3181        if (!rst_n) begin
3182            light_x_local <= '0;
3183            light_y_local <= '0;
3184            light_z_local <= '0;
3185        end else begin
3186            light_x_local <= in_light_x;
3187            light_y_local <= in_light_y;
3188            light_z_local <= in_light_z;
3189        end
3190    end
3191
3192    // 1-cycle local relay of the scene struct, mirroring the existing
3193    // broadcast-relay pattern (replaces the prior cone_local /
3194    // cone_color_local / cone_norm_factor_local / vu_local quartet).
3195    scene_t                 scene_local;
3196    logic signed [WORD-1:0] K_const_local;
3197    always_ff @(posedge clk or negedge rst_n) begin
3198        if (!rst_n) begin
3199            scene_local   <= '0;
3200            K_const_local <= '0;
3201        end else begin
3202            scene_local   <= in_scene;
3203            K_const_local <= in_K_const;
3204        end
3205    end
3206
3207    logic at_load_tick_r;
3208    always_ff @(posedge clk or negedge rst_n) begin
3209        if (!rst_n) at_load_tick_r <= 1'b0;
3210        else        at_load_tick_r <= (tick == II[5:0] - 1);
3211    end
3212    assign at_load_tick = at_load_tick_r;
3213
3214    pipe_baggage_t in_bag;
3215    assign in_bag.sv   = load_in;
3216    assign in_bag.tag  = in_tag;
3217    assign in_bag.meta = in_meta;
3218
3219    // ---- Stage A: RAY GEN ----
3220    logic signed [WORD-1:0]  raygen_raw_x, raygen_raw_y, raygen_raw_z, raygen_mag_sq;
3221    logic signed [WORD-1:0]  raygen_a_cone;
3222    logic signed [WORD-1:0]  raygen_ox, raygen_oy, raygen_oz;
3223    pipe_baggage_t           raygen_bag;
3224    rtl_ray_gen #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .WIDTH(WIDTH), .HEIGHT(HEIGHT), .TAG_W(TAG_W)) u_raygen (
3225        .*, .in_scene(scene_local)
3226    );
3227
3228    // ---- Stage B: ISECT PREP ----
3229    logic signed [WORD-1:0]  isect_dx, isect_dy, isect_dz;
3230    logic signed [WORD-1:0]  isect_ox, isect_oy, isect_oz;
3231    logic signed [WORD-1:0]  isect_inv_sa;
3232    logic signed [WORD-1:0]  isect_s0_nb, isect_s0_dsq;
3233    logic                    isect_s0_nh;
3234    logic signed [WORD-1:0]  isect_s1_nb, isect_s1_dsq;
3235    logic                    isect_s1_nh;
3236    logic signed [WORD-1:0]  isect_inv_pd;
3237    logic signed [WORD-1:0]  isect_sa;
3238    logic signed [WORD-1:0]  isect_cone_nb, isect_cone_dsq;
3239    logic                    isect_cone_nh;
3240    logic signed [WORD-1:0]  isect_inv_a_cone;
3241    logic signed [WORD-1:0]  isect_cone_dx, isect_cone_dy, isect_cone_dz;
3242    pipe_baggage_t           isect_bag;
3243    rtl_isect_prep #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_isect (
3244        .*, .in_scene(scene_local)
3245    );
3246
3247    // ---- Stage C: HIT RESOLVE (isect-resolve + hit-point/normal-prep fused) ----
3248    logic signed [WORD-1:0]  hit_hx, hit_hy, hit_hz;
3249    logic signed [WORD-1:0]  hit_dfx, hit_dfy, hit_dfz;
3250    logic signed [WORD-1:0]  hit_tlx, hit_tly, hit_tlz;
3251    logic signed [WORD-1:0]  hit_cr, hit_cg, hit_cb;
```

```
    logic [2:0]              hit_hit_type;
    logic signed [WORD-1:0]  hit_dmag;
    logic signed [WORD-1:0]  hit_dx, hit_dy, hit_dz;          // M2c+: parent d pass-through
    pipe_baggage_t           hit_bag;
    rtl_hit_resolve #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_hit (
        .*, .in_scene(scene_local)
    );

    // ---- Stage D: NORMAL INV ----
    logic signed [WORD-1:0]  norm_hx, norm_hy, norm_hz;
    logic signed [WORD-1:0]  norm_inv_dm;
    logic signed [WORD-1:0]  norm_dfx, norm_dfy, norm_dfz;
    logic signed [WORD-1:0]  norm_tlx, norm_tly, norm_tlz;
    logic signed [WORD-1:0]  norm_cr, norm_cg, norm_cb;
    logic [2:0]              norm_hit_type;
    logic signed [WORD-1:0]  norm_dx, norm_dy, norm_dz;
    pipe_baggage_t           norm_bag;
    rtl_normal_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_norm (.*);

    // ---- Stage E: REFLECT DIR + LIGHT PREP ----
    logic signed [WORD-1:0]  refl_hx, refl_hy, refl_hz;
    logic signed [WORD-1:0]  refl_nx, refl_ny, refl_nz;
    logic signed [WORD-1:0]  refl_tlx, refl_tly, refl_tlz;
    logic signed [WORD-1:0]  refl_cr, refl_cg, refl_cb;
    logic [2:0]              refl_hit_type;
    logic signed [WORD-1:0]  refl_ld;
    logic signed [WORD-1:0]  refl_rx, refl_ry, refl_rz;
    pipe_baggage_t           refl_bag;
    rtl_reflect_dir #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_refl (.*);

    // ---- Stage F: LIGHT INV ----
    logic signed [WORD-1:0]  ldir_hx, ldir_hy, ldir_hz;
    logic signed [WORD-1:0]  ldir_nx, ldir_ny, ldir_nz;
    logic signed [WORD-1:0]  ldir_tlx, ldir_tly, ldir_tlz;
    logic signed [WORD-1:0]  ldir_inv_ld;
    logic signed [WORD-1:0]  ldir_ld;
    logic signed [WORD-1:0]  ldir_cr, ldir_cg, ldir_cb;
    logic [2:0]              ldir_hit_type;
    logic signed [WORD-1:0]  ldir_rx, ldir_ry, ldir_rz;
    pipe_baggage_t           ldir_bag;
    rtl_light_dir_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_ldir (.*);

    // ---- Stage G: SHADOW TEST ----
    logic signed [WORD-1:0]  shdw_hx, shdw_hy, shdw_hz;
    logic signed [WORD-1:0]  shdw_nx, shdw_ny, shdw_nz;
    logic signed [WORD-1:0]  shdw_lx, shdw_ly, shdw_lz;
    logic signed [WORD-1:0]  shdw_cr, shdw_cg, shdw_cb;
    logic [2:0]              shdw_hit_type;
    logic                    shdw_in_shadow;
    logic signed [WORD-1:0]  shdw_rx, shdw_ry, shdw_rz;
    pipe_baggage_t           shdw_bag;
    rtl_shadow_test #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_shdw (
        .*, .in_scene(scene_local)
    );

    // ---- Stage H: SHADE ----
    // H's outputs surface hit + refl + hit_type so rtl_pipe's emit register
    // can latch them on the drop edge for the spawn router.
    logic signed [WORD-1:0]  shade_cr, shade_cg, shade_cb;
    logic signed [WORD-1:0]  shade_hx, shade_hy, shade_hz;
    logic signed [WORD-1:0]  shade_rx, shade_ry, shade_rz;
    logic [2:0]              shade_hit_type;
    pipe_baggage_t           shade_bag;
    rtl_shade_blend #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_shade (.*);

    // Emit registers: capture color + spawn side-band at t=15
    pipe_baggage_t          emit_bag;
    logic signed [WORD-1:0] emit_cr, emit_cg, emit_cb;
    logic signed [WORD-1:0] emit_hx, emit_hy, emit_hz;
    logic signed [WORD-1:0] emit_rx, emit_ry, emit_rz;
    logic [2:0]             emit_hit_type;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            emit_bag <= '0;
```

```
            emit_cr <= '0; emit_cg <= '0; emit_cb <= '0;
            emit_hx <= '0; emit_hy <= '0; emit_hz <= '0;
            emit_rx <= '0; emit_ry <= '0; emit_rz <= '0;
            emit_hit_type <= HIT_TYPE_SKY;
        end else if (tick == 6'd15) begin
            emit_bag <= shade_bag;
            emit_cr <= shade_cr;
            emit_cg <= shade_cg;
            emit_cb <= shade_cb;
            emit_hx <= shade_hx; emit_hy <= shade_hy; emit_hz <= shade_hz;
            emit_rx <= shade_rx; emit_ry <= shade_ry; emit_rz <= shade_rz;
            emit_hit_type <= shade_hit_type;
        end else begin
            emit_bag.sv <= 1'b0;
        end
    end

    assign drop_out    = emit_bag.sv;
    assign out_tag     = emit_bag.tag[TAG_W-1:0];
    assign out_color_r = emit_cr;
    assign out_color_g = emit_cg;
    assign out_color_b = emit_cb;
    assign out_meta    = emit_bag.meta;
    assign out_hit_x   = emit_hx;
    assign out_hit_y   = emit_hy;
    assign out_hit_z   = emit_hz;
    assign out_refl_x  = emit_rx;
    assign out_refl_y  = emit_ry;
    assign out_refl_z  = emit_rz;
    assign out_hit_type = emit_hit_type;

    // Spawn decision: combinational. reflect_shift_for_hit is read from
    // the registered scene_local so it matches the rest of stage-K's data
    // path (and gives Quartus a placement handle near the consumer).
    logic [4:0] reflect_shift_for_hit;
    logic       spawn_decision;
    always_comb begin
        unique case (emit_hit_type)
            HIT_TYPE_S0:   reflect_shift_for_hit = scene_local.s0.reflect_shift;
            HIT_TYPE_S1:   reflect_shift_for_hit = scene_local.s1.reflect_shift;
            HIT_TYPE_CONE: reflect_shift_for_hit = scene_local.cone.reflect_shift;
            default:       reflect_shift_for_hit = '0;
        endcase
    end
    assign spawn_decision = (emit_bag.meta.depth < max_depth_r) &&
                            ((emit_hit_type == HIT_TYPE_S0)   ||
                             (emit_hit_type == HIT_TYPE_S1)   ||
                             (emit_hit_type == HIT_TYPE_CONE)) &&
                            (reflect_shift_for_hit != '0) &&
                            (({1'b0, emit_bag.meta.total_shift} +
                              {1'b0, reflect_shift_for_hit}) < 6'd8);
    assign out_spawn_push           = emit_bag.sv & spawn_decision;
    assign out_spawn_meta.depth       = emit_bag.meta.depth + 3'd1;
    assign out_spawn_meta.excluded    = emit_hit_type;
    assign out_spawn_meta.total_shift = emit_bag.meta.total_shift + reflect_shift_for_hit;
endmodule

// ============================================================================
// basis_compute - shared 4c-latency camera basis transform. One instance per
// rtl_batch_pipe (was K replicas inside each per-pipe stage A; the K-fold
// redundancy was unnecessary since all pipes consume the same basis per
// frame). 5 dedicated DSPs (right·px + up·py + fwd, with right.y=0 for no
// roll).
// ============================================================================
module basis_compute #(
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int WIDTH     = 480,
    parameter int HEIGHT    = 360
)(
    input  logic                        clk, rst_n,                       // clock, async neg-edge reset
    input  logic                        kick,                              // 1c pulse: latch pix_*_in, begin 5-
cycle compute
    input  logic [$clog2(WIDTH)-1:0]   pix_x_in,                           // pixel column (0..WIDTH-1)
```

```
    input  logic [$clog2(HEIGHT)-1:0]  pix_y_in,                          // pixel row    (0..HEIGHT-1)
    input  logic signed [WORD-1:0]     right_x_in, right_z_in,            // camera basis right.{x,z}; right.y=0 by no-roll convention
    input  logic signed [WORD-1:0]     up_x_in, up_y_in, up_z_in,         // camera basis up
    input  logic signed [WORD-1:0]     fwd_x_in, fwd_y_in, fwd_z_in,      // camera basis fwd
    output logic signed [WORD-1:0]     raw_x_out, raw_y_out, raw_z_out    // un-normalized ray direction = right·px + up·py + fwd; valid 5c after kick
);
    localparam signed [WORD-1:0] ONE    = 1 <<< FRAC_BITS;
    localparam signed [WORD-1:0] HALF_W = ONE / 2;
    localparam signed [WORD-1:0] HALF_H = (HALF_W * HEIGHT) / WIDTH;

    localparam int RECIP_SHIFT = 24;
    localparam longint INV_W_Q = ((longint'(2 * HALF_W)) <<< RECIP_SHIFT) / WIDTH;
    localparam longint INV_H_Q = ((longint'(2 * HALF_H)) <<< RECIP_SHIFT) / HEIGHT;
    localparam signed [31:0] INV_W = INV_W_Q[31:0];
    localparam signed [31:0] INV_H = INV_H_Q[31:0];

    logic [$clog2(WIDTH)-1:0]   pix_x_r;
    logic [$clog2(HEIGHT)-1:0]  pix_y_local_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            pix_x_r       <= '0;
            pix_y_local_r <= '0;
        end else if (kick) begin
            pix_x_r       <= pix_x_in;
            pix_y_local_r <= pix_y_in;
        end
    end

    localparam int PXC_W = 14;
    logic signed [WORD-1:0]     px_s, py_s;
    logic signed [WORD+32-1:0]  px_prod, py_prod;
    logic signed [PXC_W-1:0]    px_c_w, py_c_w;
    assign px_s    = signed'({{(WORD-$clog2(WIDTH)){1'b0}},  pix_x_r});
    assign py_s    = signed'({{(WORD-$clog2(HEIGHT)){1'b0}}, pix_y_local_r});
    assign px_prod = px_s * INV_W;
    assign py_prod = py_s * INV_H;
    assign px_c_w  = PXC_W'(WORD'(px_prod >>> RECIP_SHIFT) - HALF_W);
    assign py_c_w  = PXC_W'(HALF_H - WORD'(py_prod >>> RECIP_SHIFT));

    logic signed [PXC_W-1:0]    px_c_r, py_c_r;
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            px_c_r <= '0;
            py_c_r <= '0;
        end else begin
            px_c_r <= px_c_w;
            py_c_r <= py_c_w;
        end
    end

    logic signed [WORD-1:0] m_rx_px, m_rz_px, m_ux_py, m_uy_py, m_uz_py;
    fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_rx_px (
        .clk(clk), .a(right_x_in), .b(px_c_r), .result(m_rx_px));
    fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_rz_px (
        .clk(clk), .a(right_z_in), .b(px_c_r), .result(m_rz_px));
    fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_ux_py (
        .clk(clk), .a(up_x_in),    .b(py_c_r), .result(m_ux_py));
    fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_uy_py (
        .clk(clk), .a(up_y_in),    .b(py_c_r), .result(m_uy_py));
    fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_uz_py (
        .clk(clk), .a(up_z_in),    .b(py_c_r), .result(m_uz_py));

    logic signed [WORD-1:0] rx_px_r, rz_px_r, ux_py_r, uy_py_r, uz_py_r;
    logic signed [WORD-1:0] raw_x_r, raw_y_r, raw_z_r;

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            rx_px_r <= '0; rz_px_r <= '0;
            ux_py_r <= '0; uy_py_r <= '0; uz_py_r <= '0;
            raw_x_r <= '0; raw_y_r <= '0; raw_z_r <= '0;
        end else begin
            rx_px_r <= m_rx_px;
```

```
            rz_px_r <= m_rz_px;
            ux_py_r <= m_ux_py;
            uy_py_r <= m_uy_py;
            uz_py_r <= m_uz_py;
            raw_x_r <= rx_px_r + ux_py_r + fwd_x_in;
            raw_y_r <= uy_py_r + fwd_y_in;
            raw_z_r <= rz_px_r + uz_py_r + fwd_z_in;
        end
    end

    assign raw_x_out = raw_x_r;
    assign raw_y_out = raw_y_r;
    assign raw_z_out = raw_z_r;
endmodule

// ============================================================================
// rtl_batch_pipe - K parallel rtl_pipe instances + issue arbiter + per-pipe
// spawn slots + BRAM write port. Pipes have staggered PHASEs (II·i/K) so at
// most one is at tick=0 each cycle. Issue arbiter picks the pipe at t=0 and
// hands it either a primary (next pix_base + issue_count) or its own pipe's
// spawn slot if valid. Each pipe's drop fires once per ray at its own t=15.
// Spawn router writes reflection children with depth+1 / total_shift
// accumulated into the dropping pipe's own slot; primaries retire to BRAM
// via the wr_en/wr_addr/wr_data port.
// ============================================================================
module rtl_batch_pipe
    import raytracer_pkg::*;
#(
    parameter int K         = 4,
    parameter int BATCH     = 120,
    parameter int FRAC_BITS = 13,
    parameter int WORD      = 27,
    parameter int WIDTH     = 480,
    parameter int HEIGHT    = 360
)(
    input  logic                        clk,                            // clock
    input  logic                        rst_n,                          // active-low async reset
    input  logic                        start,                          // 1c pulse: latch all inputs, kick batch
    input  logic [$clog2(WIDTH)-1:0]    pixel_x,                        // batch's starting pixel-x (= pix_base)
    input  logic [$clog2(HEIGHT)-1:0]   pixel_y,                        // batch's row (pix_y_r)
    input  logic signed [WORD-1:0]      light_x, light_y, light_z,      // light position (latched at start)
    input  scene_t                      scene,                          // bundled primitives + floor (latched at
start)
    input  logic [2:0]                  max_depth,                      // reflection recursion cap (0 disables)
    // cam_y is a constant from the wrapper (no Y-translation CSR).
    // right.y is hardcoded 0 (no roll) so it's not transmitted.
    input  logic signed [WORD-1:0]      cam_x, cam_y, cam_z,            // camera origin (cam_y locked at 1.5 by
wrapper)
    input  logic signed [WORD-1:0]      right_x, right_z,               // camera basis right.{x,z}; right.y=0 (no
 roll)
    input  logic signed [WORD-1:0]      up_x, up_y, up_z,               // camera basis up
    input  logic signed [WORD-1:0]      fwd_x, fwd_y, fwd_z,            // camera basis fwd
    input  logic signed [WORD-1:0]      K_const,                        // |light|²·63/64 - sun-billboard
threshold
    output logic                        done,                           // 1c pulse: batch complete (all pixels
emitted)
    // BRAM write port: one indexed write per cycle when a pipe drops a
    // completed lane (1c spacing guaranteed by one-hot issue scheduling).
    output logic                        wr_en,                          // write enable to color_mem (in wrapper)
    output logic [$clog2(BATCH)-1:0]    wr_addr,                        // lane tag = pixel-within-batch index
    output logic [23:0]                 wr_data                         // saturated 8-bit BGR pixel color
);
    localparam int II    = 43;
    localparam int TAG_W = (BATCH <= 1) ? 1 : $clog2(BATCH);

    logic [$clog2(WIDTH)-1:0]   pix_base;
    logic [$clog2(HEIGHT)-1:0]  pix_y_r;
    logic signed [WORD-1:0]     in_light_x, in_light_y, in_light_z;
    scene_t                     in_scene;
    logic [2:0]                 max_depth_r;
    logic signed [WORD-1:0]     cam_x_r, cam_y_r, cam_z_r;
    logic signed [WORD-1:0]     right_x_r, right_z_r;
    logic signed [WORD-1:0]     up_x_r, up_y_r, up_z_r;
    logic signed [WORD-1:0]     fwd_x_r, fwd_y_r, fwd_z_r;
```

```
    logic signed [WORD-1:0]     in_K_const;

    // Per-pipe broadcast relays. `(* preserve, dont_merge *)` keeps the
    // K copies physically distinct so Quartus can place each one near its
    // consumer instead of folding them into a single FF with high fanout.
    (* preserve, dont_merge *) logic signed [WORD-1:0]     in_light_x_p [K];
    (* preserve, dont_merge *) logic signed [WORD-1:0]     in_light_y_p [K];
    (* preserve, dont_merge *) logic signed [WORD-1:0]     in_light_z_p [K];
    (* preserve, dont_merge *) logic signed [WORD-1:0]     in_K_const_p [K];

    (* preserve, dont_merge *) scene_t                     in_scene_p   [K];

    // Shared basis_compute outputs (single instance below; broadcast to
    // every pipe's stage A).
    logic signed [WORD-1:0]     in_basis_raw_x, in_basis_raw_y, in_basis_raw_z;

    logic [$clog2(BATCH+1)-1:0] issue_count;
    logic [$clog2(BATCH+1)-1:0] complete_count;
    logic                       busy;

    // K-pipe issue + drop side-bands.
    logic                       pipe_load     [K];
    logic [$clog2(WIDTH)-1:0]   pipe_pix_x    [K];
    logic [TAG_W-1:0]           pipe_tag_in   [K];
    ray_meta_t                  pipe_meta_in  [K];

    logic                       pipe_use_override [K];
    logic signed [WORD-1:0]     pipe_orig_x       [K];
    logic signed [WORD-1:0]     pipe_orig_y       [K];
    logic signed [WORD-1:0]     pipe_orig_z       [K];
    logic signed [WORD-1:0]     pipe_dir_x        [K];
    logic signed [WORD-1:0]     pipe_dir_y        [K];
    logic signed [WORD-1:0]     pipe_dir_z        [K];
    logic                       pipe_drop     [K];
    logic [TAG_W-1:0]           pipe_tag_out  [K];
    logic signed [WORD-1:0]     pipe_color_r  [K];
    logic signed [WORD-1:0]     pipe_color_g  [K];
    logic signed [WORD-1:0]     pipe_color_b  [K];
    ray_meta_t                  pipe_meta_out [K];
    // Spawn side-band, stable on the drop edge.
    logic signed [WORD-1:0]     pipe_out_hit_x   [K];
    logic signed [WORD-1:0]     pipe_out_hit_y   [K];
    logic signed [WORD-1:0]     pipe_out_hit_z   [K];
    logic signed [WORD-1:0]     pipe_out_refl_x  [K];
    logic signed [WORD-1:0]     pipe_out_refl_y  [K];
    logic signed [WORD-1:0]     pipe_out_refl_z  [K];
    logic [2:0]                 pipe_out_hit_type [K];
    logic                       pipe_out_spawn_push [K];
    ray_meta_t                  pipe_out_spawn_meta [K];
    logic                       pipe_at_load  [K];

    genvar gi;
    generate
        for (gi = 0; gi < K; gi++) begin : g_pipe
            localparam int PHASE = (II * gi) / K;
            rtl_pipe #(
                .FRAC_BITS(FRAC_BITS), .WORD(WORD),
                .WIDTH(WIDTH), .HEIGHT(HEIGHT),
                .TAG_W(TAG_W), .II(II), .PHASE(PHASE)
            ) u_pipe (
                .*,
                .in_light_x          (in_light_x_p[gi]),
                .in_light_y          (in_light_y_p[gi]),
                .in_light_z          (in_light_z_p[gi]),
                .in_K_const          (in_K_const_p[gi]),
                .in_scene            (in_scene_p[gi]),
                .load_in       (pipe_load[gi]),
                .in_tag        (pipe_tag_in[gi]),
                .in_meta       (pipe_meta_in[gi]),
                .in_use_override(pipe_use_override[gi]),
                .in_orig_x     (pipe_orig_x[gi]),
                .in_orig_y     (pipe_orig_y[gi]),
                .in_orig_z     (pipe_orig_z[gi]),
                .in_dir_x      (pipe_dir_x[gi]),
```

```
                .in_dir_y      (pipe_dir_y[gi]),
                .in_dir_z      (pipe_dir_z[gi]),
                .drop_out      (pipe_drop[gi]),
                .out_tag       (pipe_tag_out[gi]),
                .out_color_r   (pipe_color_r[gi]),
                .out_color_g   (pipe_color_g[gi]),
                .out_color_b   (pipe_color_b[gi]),
                .out_meta      (pipe_meta_out[gi]),
                .out_hit_x     (pipe_out_hit_x[gi]),
                .out_hit_y     (pipe_out_hit_y[gi]),
                .out_hit_z     (pipe_out_hit_z[gi]),
                .out_refl_x    (pipe_out_refl_x[gi]),
                .out_refl_y    (pipe_out_refl_y[gi]),
                .out_refl_z    (pipe_out_refl_z[gi]),
                .out_hit_type  (pipe_out_hit_type[gi]),
                .out_spawn_push(pipe_out_spawn_push[gi]),
                .out_spawn_meta(pipe_out_spawn_meta[gi]),
                .at_load_tick  (pipe_at_load[gi])
            );
        end
    endgenerate

    // ---- Per-pipe spawn slots ----------------------------------------
    //
    // Each pipe owns one single-entry spawn slot. When stage H of pipe i
    // emits a drop for a ray that hit a sphere/cone with depth < max_depth,
    // the spawn router writes into spawn_slot[i] with the child entry
    // (same tag, parent hit point as origin, reflection direction, and
    // meta with depth+1, excluded=parent's hit_type, total_shift updated).
    // The issue arbiter consumes pipe i's slot at pipe i's load_tick,
    // preferring the slot over a new primary so chains progress promptly.
    //
    // Single-entry is sufficient because: (a) stage H of each pipe emits
    // at most one drop per II window (the drop is a 1-cycle pulse at a
    // fixed tick), and (b) push and pop on the same pipe occur at
    // different ticks of the pipe's local II schedule (push at the stage-K
    // emit tick, pop at the load_tick), so they never collide on the same
    // pipe in the same cycle.
    typedef struct packed {
        logic [TAG_W-1:0]      tag;
        logic signed [WORD-1:0] orig_x, orig_y, orig_z;
        logic signed [WORD-1:0] dir_x, dir_y, dir_z;
        ray_meta_t             meta;
    } spawn_entry_t;

    spawn_entry_t              spawn_slot       [K];
    logic                      spawn_slot_valid [K];

    logic                      spawn_push       [K];
    spawn_entry_t              spawn_push_data  [K];

    logic              issue_fire;
    logic              issue_is_spawn;
    logic [$clog2(K):0] issue_pidx;
    always_comb begin
        issue_fire     = 1'b0;
        issue_is_spawn = 1'b0;
        issue_pidx     = '0;
        for (int p = 0; p < K; p++) begin
            if (!issue_fire && pipe_at_load[p] && busy) begin
                if (spawn_slot_valid[p]) begin
                    issue_fire     = 1'b1;
                    issue_is_spawn = 1'b1;
                    issue_pidx     = p[$clog2(K):0];
                end else if (issue_count < BATCH) begin
                    issue_fire     = 1'b1;
                    issue_is_spawn = 1'b0;
                    issue_pidx     = p[$clog2(K):0];
                end
            end
        end
        for (int p = 0; p < K; p++) begin
            automatic logic use_spawn_p = issue_fire && (p == issue_pidx) && issue_is_spawn;
            pipe_load[p]         = issue_fire && (p == issue_pidx);
```

```
            pipe_pix_x[p]        = pix_base + $clog2(WIDTH)'(issue_count);
            pipe_tag_in[p]       = use_spawn_p ? spawn_slot[p].tag
                                               : issue_count[TAG_W-1:0];
            pipe_meta_in[p]      = use_spawn_p ? spawn_slot[p].meta : '0;
            pipe_use_override[p] = use_spawn_p;
            pipe_orig_x[p]       = use_spawn_p ? spawn_slot[p].orig_x : cam_x_r;
            pipe_orig_y[p]       = use_spawn_p ? spawn_slot[p].orig_y : cam_y_r;
            pipe_orig_z[p]       = use_spawn_p ? spawn_slot[p].orig_z : cam_z_r;
            pipe_dir_x[p]        = spawn_slot[p].dir_x;
            pipe_dir_y[p]        = spawn_slot[p].dir_y;
            pipe_dir_z[p]        = spawn_slot[p].dir_z;
        end
    end

    logic bm_kick;
    assign bm_kick = issue_fire && !issue_is_spawn;
    basis_compute #(
        .FRAC_BITS(FRAC_BITS), .WORD(WORD),
        .WIDTH(WIDTH),         .HEIGHT(HEIGHT)
    ) u_basis (
        .clk(clk), .rst_n(rst_n),
        .kick(bm_kick),
        .pix_x_in(pipe_pix_x[issue_pidx]),
        .pix_y_in(pix_y_r),
        .right_x_in(right_x_r), .right_z_in(right_z_r),
        .up_x_in(up_x_r),       .up_y_in(up_y_r),       .up_z_in(up_z_r),
        .fwd_x_in(fwd_x_r),     .fwd_y_in(fwd_y_r),     .fwd_z_in(fwd_z_r),
        .raw_x_out(in_basis_raw_x),   .raw_y_out(in_basis_raw_y),   .raw_z_out(in_basis_raw_z)
    );

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (int p = 0; p < K; p++) spawn_slot_valid[p] <= 1'b0;
        end else if (start) begin
            for (int p = 0; p < K; p++) spawn_slot_valid[p] <= 1'b0;
        end else begin
            for (int p = 0; p < K; p++) begin
                if (spawn_push[p])
                    spawn_slot_valid[p] <= 1'b1;
                else if (issue_fire && issue_is_spawn && (issue_pidx == p))
                    spawn_slot_valid[p] <= 1'b0;
            end
        end
    end
    always_ff @(posedge clk) begin
        for (int p = 0; p < K; p++)
            if (spawn_push[p]) spawn_slot[p] <= spawn_push_data[p];
    end

    localparam signed [WORD-1:0] COLOR_ONE = 1 <<< FRAC_BITS;
    function automatic logic [7:0] to_byte(input logic signed [WORD-1:0] v);
        return v[FRAC_BITS] ? 8'hff : v[FRAC_BITS-1 -: 8];
    endfunction

    // Shared accumulator BRAM. Per-tag slot holds {pending, rgb sum}.
    typedef struct packed {
        logic        pending;
        logic [15:0] r, g, b;     // 16-bit per channel: covers  MAX_DEPTH7
                                  // with power-of-2 attenuation per bounce.
    } lane_acc_t;

    localparam lane_acc_t LANE_ACC_INIT =
        '{pending: 1'b1, r: 16'd0, g: 16'd0, b: 16'd0};

    logic              any_drop_w;
    logic [TAG_W-1:0]  drop_tag_w;
    logic signed [WORD-1:0] drop_color_r_w, drop_color_g_w, drop_color_b_w;
    ray_meta_t         drop_meta_w;
    logic [2:0]        drop_hit_type_w;
    always_comb begin
        any_drop_w      = 1'b0;
        drop_tag_w      = '0;
        drop_color_r_w  = '0;
        drop_color_g_w  = '0;
```

```
        drop_color_b_w  = '0;
        drop_meta_w     = '0;
        drop_hit_type_w = HIT_TYPE_SKY;
        for (int p = 0; p < K; p++) begin
            if (pipe_drop[p]) begin
                any_drop_w      = 1'b1;
                drop_tag_w      = pipe_tag_out[p];
                drop_color_r_w  = pipe_color_r[p];
                drop_color_g_w  = pipe_color_g[p];
                drop_color_b_w  = pipe_color_b[p];
                drop_meta_w     = pipe_meta_out[p];
                drop_hit_type_w = pipe_out_hit_type[p];
            end
        end
    end

    // Init on primary issue only - spawn children must not wipe the
    // parent's running sum.
    logic            init_lane_w;
    logic [TAG_W-1:0] init_tag_w;
    always_comb begin
        init_lane_w = issue_fire && !issue_is_spawn;
        init_tag_w  = issue_count[TAG_W-1:0];
    end

    (* ramstyle = "no_rw_check, M10K" *)
    lane_acc_t acc_mem [BATCH];

    // 1-cycle pipeline of drop info (covers BRAM read latency). The
    // always_ff below latches the _w signals into _d1 regs on cycle T and
    // fires the M10K read; the always_comb blocks consume the _d1 regs +
    // acc_rd_d1 on cycle T+1 to produce acc_new, which the same always_ff
    // writes back into acc_mem.
    logic             drop_d1;
    logic [TAG_W-1:0]  drop_tag_d1;
    logic [7:0]        drop_byte_r_d1, drop_byte_g_d1, drop_byte_b_d1;
    ray_meta_t         drop_meta_d1;
    logic [2:0]        drop_hit_type_d1;
    lane_acc_t         acc_rd_d1;
    lane_acc_t         acc_new;
    logic [4:0]        drop_reflect_shift_d1;
    logic              is_spawn_d1;
    logic [7:0]        weighted_r, weighted_g, weighted_b;

    always_ff @(posedge clk) begin
        drop_d1            <= any_drop_w;
        drop_tag_d1        <= drop_tag_w;
        drop_byte_r_d1     <= to_byte(drop_color_r_w);
        drop_byte_g_d1     <= to_byte(drop_color_g_w);
        drop_byte_b_d1     <= to_byte(drop_color_b_w);
        drop_meta_d1       <= drop_meta_w;
        drop_hit_type_d1   <= drop_hit_type_w;

        if (any_drop_w)
            acc_rd_d1 <= acc_mem[drop_tag_w];

        if (init_lane_w)
            acc_mem[init_tag_w] <= LANE_ACC_INIT;
        else if (drop_d1)
            acc_mem[drop_tag_d1] <= acc_new;
    end

    always_comb begin
        unique case (drop_hit_type_d1)
            // Accumulator runs at the merge point - uses the wrapper-level
            // single-FF batch-state copy (in_scene), not the per-pipe relay.
            HIT_TYPE_S0:   drop_reflect_shift_d1 = in_scene.s0.reflect_shift;
            HIT_TYPE_S1:   drop_reflect_shift_d1 = in_scene.s1.reflect_shift;
            HIT_TYPE_CONE: drop_reflect_shift_d1 = in_scene.cone.reflect_shift;
            default:       drop_reflect_shift_d1 = '0;
        endcase
    end

    assign is_spawn_d1 = (drop_meta_d1.depth < max_depth_r) &&
```

```
                       ((drop_hit_type_d1 == HIT_TYPE_S0) ||
                        (drop_hit_type_d1 == HIT_TYPE_S1) ||
                        (drop_hit_type_d1 == HIT_TYPE_CONE)) &&
                       (drop_reflect_shift_d1 != '0) &&
                       (({1'b0, drop_meta_d1.total_shift} +
                         {1'b0, drop_reflect_shift_d1}) < 6'd8);

    always_comb begin
        if (drop_meta_d1.total_shift >= 5'd8) begin
            weighted_r = 8'd0;
            weighted_g = 8'd0;
            weighted_b = 8'd0;
        end else begin
            weighted_r = drop_byte_r_d1 >> drop_meta_d1.total_shift;
            weighted_g = drop_byte_g_d1 >> drop_meta_d1.total_shift;
            weighted_b = drop_byte_b_d1 >> drop_meta_d1.total_shift;
        end
    end

    always_comb begin
        acc_new.r       = acc_rd_d1.r + 16'(weighted_r);
        acc_new.g       = acc_rd_d1.g + 16'(weighted_g);
        acc_new.b       = acc_rd_d1.b + 16'(weighted_b);
        acc_new.pending = is_spawn_d1;
    end

    // Single emit pulse - fires on chain termination.
    logic              acc_emit_w;
    logic [TAG_W-1:0]  acc_emit_tag_w;
    logic [15:0]       acc_emit_r_w, acc_emit_g_w, acc_emit_b_w;
    assign acc_emit_w     = drop_d1 && !acc_new.pending;
    assign acc_emit_tag_w = drop_tag_d1;
    assign acc_emit_r_w   = acc_new.r;
    assign acc_emit_g_w   = acc_new.g;
    assign acc_emit_b_w   = acc_new.b;

    // Spawn router: per-pipe drivers
    always_comb begin
        for (int p = 0; p < K; p++) begin
            spawn_push[p]      = pipe_out_spawn_push[p];
            spawn_push_data[p] = '0;
            if (pipe_out_spawn_push[p]) begin
                spawn_push_data[p].tag    = pipe_tag_out[p];
                spawn_push_data[p].orig_x = pipe_out_hit_x[p];
                spawn_push_data[p].orig_y = pipe_out_hit_y[p];
                spawn_push_data[p].orig_z = pipe_out_hit_z[p];
                spawn_push_data[p].dir_x  = pipe_out_refl_x[p];
                spawn_push_data[p].dir_y  = pipe_out_refl_y[p];
                spawn_push_data[p].dir_z  = pipe_out_refl_z[p];
                spawn_push_data[p].meta   = pipe_out_spawn_meta[p];
            end
        end
    end

    // 16-bit accumulator → 8-bit channel with high-byte saturation.
    function automatic logic [7:0] sat_byte(input logic [15:0] v);
        return v[15:8] != 8'd0 ? 8'hff : v[7:0];
    endfunction

    // Completion counts on chain-terminal emits, not raw drops, so spawn
    // children don't double-increment.
    logic any_emit;
    assign any_emit = acc_emit_w;

    // -------- Batch scene state latch --------
    // Captures all batch-stable broadcasts on `start` and holds them for
    // the duration of the batch. SW invariant: CSRs are only rewritten
    // between batches (WAIT_FRAME → write CSRs → FRAME_START), so once
    // these latch they don't change until the next start.
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            pix_base <= '0; pix_y_r <= '0;
            in_light_x <= '0; in_light_y <= '0; in_light_z <= '0;
            in_scene   <= '0;
```

```
            max_depth_r <= '0;
            cam_x_r <= '0; cam_y_r <= '0; cam_z_r <= '0;
            right_x_r <= '0; right_z_r <= '0;
            up_x_r <= '0; up_y_r <= '0; up_z_r <= '0;
            fwd_x_r <= '0; fwd_y_r <= '0; fwd_z_r <= '0;
            in_K_const <= '0;
        end else if (start) begin
            pix_base <= pixel_x;
            pix_y_r  <= pixel_y;
            in_light_x <= light_x; in_light_y <= light_y; in_light_z <= light_z;
            in_scene   <= scene;
            max_depth_r <= max_depth;
            cam_x_r <= cam_x; cam_y_r <= cam_y; cam_z_r <= cam_z;
            right_x_r <= right_x; right_z_r <= right_z;
            up_x_r <= up_x; up_y_r <= up_y; up_z_r <= up_z;
            fwd_x_r <= fwd_x; fwd_y_r <= fwd_y; fwd_z_r <= fwd_z;
            in_K_const <= K_const;
        end
    end

    // -------- Batch lifecycle counters --------
    // busy asserts on start, deasserts when complete_count reaches BATCH.
    // issue_count = primaries issued (spawns reuse parent's tag and must
    // not increment). complete_count = chain-terminal emits (any_emit).
    // done = 1-cycle pulse on the BATCH-th completion.
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            issue_count    <= '0;
            complete_count <= '0;
            busy           <= 1'b0;
            done           <= 1'b0;
        end else begin
            done <= 1'b0;
            if (start) begin
                issue_count    <= '0;
                complete_count <= '0;
                busy           <= 1'b1;
            end else if (busy) begin
                if (issue_fire && !issue_is_spawn) issue_count <= issue_count + 1;
                if (any_emit) complete_count <= complete_count + 1;
                if (any_emit && (complete_count == ($clog2(BATCH+1))'(BATCH - 1))) begin
                    done <= 1'b1;
                    busy <= 1'b0;
                end
            end
        end
    end

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (int i = 0; i < K; i++) begin
                in_light_x_p[i] <= '0;
                in_light_y_p[i] <= '0;
                in_light_z_p[i] <= '0;
                in_K_const_p[i] <= '0;
                in_scene_p[i]   <= '0;
            end
        end else begin
            for (int i = 0; i < K; i++) begin
                in_light_x_p[i] <= in_light_x;
                in_light_y_p[i] <= in_light_y;
                in_light_z_p[i] <= in_light_z;
                in_K_const_p[i] <= in_K_const;
                in_scene_p[i]   <= in_scene;
            end
        end
    end

    // BRAM write on terminal chain emit. acc_emit_w is one-hot per cycle.
    always_comb begin
        wr_en   = acc_emit_w;
        wr_addr = acc_emit_tag_w;
        wr_data = {sat_byte(acc_emit_r_w),
                   sat_byte(acc_emit_g_w),
```

```
                    sat_byte(acc_emit_b_w)};
    end

`ifdef SIMULATION
    // Guard the one-drop-per-cycle invariant the BRAM write port assumes.
    int drop_count;
    always_comb begin
        drop_count = 0;
        for (int p = 0; p < K; p++) begin
            if (pipe_drop[p]) drop_count = drop_count + 1;
        end
    end
    always_ff @(posedge clk) begin
        if (rst_n && drop_count > 1)
            $error("rtl_batch_pipe: multiple simultaneous drops (single-port BRAM assumption violated)");
    end
`endif

endmodule
```

raytracer_hw.tcl

```
# TCL File Generated by Component Editor 21.1
# Tue Apr 14 17:37:35 EDT 2026
# DO NOT MODIFY


#
# raytracer "Raytracer" v1.0
#  2026.04.14.17:37:35
#
#

#
# request TCL package from ACDS 16.1
#
package require -exact qsys 16.1


#
# module raytracer
#
set_module_property DESCRIPTION ""
set_module_property NAME raytracer
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME Raytracer
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false


#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL raytracer_batch_mm
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file raytracer.sv SYSTEM_VERILOG PATH raytracer.sv TOP_LEVEL_FILE


#
# parameters
#
add_parameter N INTEGER 480 ""
set_parameter_property N DEFAULT_VALUE 480
set_parameter_property N DISPLAY_NAME N
```

```
set_parameter_property N WIDTH ""
set_parameter_property N TYPE INTEGER
set_parameter_property N UNITS None
set_parameter_property N ALLOWED_RANGES -2147483648:2147483647
set_parameter_property N DESCRIPTION ""
set_parameter_property N HDL_PARAMETER true
add_parameter FRAC_BITS INTEGER 13 ""
set_parameter_property FRAC_BITS DEFAULT_VALUE 13
set_parameter_property FRAC_BITS DISPLAY_NAME FRAC_BITS
set_parameter_property FRAC_BITS WIDTH ""
set_parameter_property FRAC_BITS TYPE INTEGER
set_parameter_property FRAC_BITS UNITS None
set_parameter_property FRAC_BITS ALLOWED_RANGES -2147483648:2147483647
set_parameter_property FRAC_BITS DESCRIPTION ""
set_parameter_property FRAC_BITS HDL_PARAMETER true
add_parameter WORD INTEGER 27 ""
set_parameter_property WORD DEFAULT_VALUE 27
set_parameter_property WORD DISPLAY_NAME WORD
set_parameter_property WORD WIDTH ""
set_parameter_property WORD TYPE INTEGER
set_parameter_property WORD UNITS None
set_parameter_property WORD ALLOWED_RANGES -2147483648:2147483647
set_parameter_property WORD DESCRIPTION ""
set_parameter_property WORD HDL_PARAMETER true
add_parameter WIDTH INTEGER 480 ""
set_parameter_property WIDTH DEFAULT_VALUE 480
set_parameter_property WIDTH DISPLAY_NAME WIDTH
set_parameter_property WIDTH WIDTH ""
set_parameter_property WIDTH TYPE INTEGER
set_parameter_property WIDTH UNITS None
set_parameter_property WIDTH ALLOWED_RANGES -2147483648:2147483647
set_parameter_property WIDTH DESCRIPTION ""
set_parameter_property WIDTH HDL_PARAMETER true
add_parameter HEIGHT INTEGER 360 ""
set_parameter_property HEIGHT DEFAULT_VALUE 360
set_parameter_property HEIGHT DISPLAY_NAME HEIGHT
set_parameter_property HEIGHT WIDTH ""
set_parameter_property HEIGHT TYPE INTEGER
set_parameter_property HEIGHT UNITS None
set_parameter_property HEIGHT ALLOWED_RANGES -2147483648:2147483647
set_parameter_property HEIGHT DESCRIPTION ""
set_parameter_property HEIGHT HDL_PARAMETER true


#
# module assignments
#
set_module_assignment embeddedsw.dts.group raytracer
set_module_assignment embeddedsw.dts.name raytracer
set_module_assignment embeddedsw.dts.vendor csee4840


#
# display items
#


#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1


#
# connection point reset
#
```

```
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset reset reset Input 1


#
# connection point avalon_slave_0
#
add_interface avalon_slave_0 avalon end
set_interface_property avalon_slave_0 addressUnits WORDS
set_interface_property avalon_slave_0 associatedClock clock
set_interface_property avalon_slave_0 associatedReset reset
set_interface_property avalon_slave_0 bitsPerSymbol 8
set_interface_property avalon_slave_0 explicitAddressSpan 0
set_interface_property avalon_slave_0 holdTime 0
set_interface_property avalon_slave_0 maximumPendingReadTransactions 0
set_interface_property avalon_slave_0 maximumPendingWriteTransactions 0
set_interface_property avalon_slave_0 readLatency 1
set_interface_property avalon_slave_0 setupTime 0
set_interface_property avalon_slave_0 timingUnits Cycles
set_interface_property avalon_slave_0 writeWaitTime 0
set_interface_property avalon_slave_0 ENABLED true
set_interface_property avalon_slave_0 EXPORT_OF ""
set_interface_property avalon_slave_0 PORT_NAME_MAP ""
set_interface_property avalon_slave_0 CMSIS_SVD_VARIABLES ""
set_interface_property avalon_slave_0 SVD_ADDRESS_GROUP ""

add_interface_port avalon_slave_0 chipselect chipselect Input 1
add_interface_port avalon_slave_0 read read Input 1
add_interface_port avalon_slave_0 write write Input 1
add_interface_port avalon_slave_0 address address Input 10
add_interface_port avalon_slave_0 writedata writedata Input 32
add_interface_port avalon_slave_0 readdata readdata Output 32
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isFlash 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isMemoryDevice 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isNonVolatileStorage 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isPrintableDevice 0


#
# connection point avalon_master_0
#
# Avalon-MM master for DMA-out. Pushes color results to a SW-programmed
# HPS DDR3 phys address via hps_0.f2h_axi_slave. qsys auto-bridges the
# Avalon master to the AXI3 slave; the bridge generates AWUSER/AWPROT/
# AWCACHE for f2h_axi_slave automatically. Pattern matches `master_secure`
# in the DE1-SoC DE1_SOC_Linux_FB reference design.
#
# Single 240-beat burst per batch (1920 B at 64-bit). The bridge splits
# this into AXI3 16-beat bursts internally to satisfy the AXI3 burst
# length limit.
add_interface avalon_master_0 avalon start
set_interface_property avalon_master_0 addressUnits SYMBOLS
set_interface_property avalon_master_0 associatedClock clock
set_interface_property avalon_master_0 associatedReset reset
set_interface_property avalon_master_0 bitsPerSymbol 8
set_interface_property avalon_master_0 burstOnBurstBoundariesOnly false
set_interface_property avalon_master_0 burstcountUnits WORDS
set_interface_property avalon_master_0 doStreamReads false
set_interface_property avalon_master_0 doStreamWrites false
set_interface_property avalon_master_0 holdTime 0
set_interface_property avalon_master_0 linewrapBursts false
set_interface_property avalon_master_0 maximumPendingReadTransactions 0
# Set to 0 instead of 1: a non-zero pending-write count requires the
# master to expose `response` + `writeresponsevalid` signals (Quartus
# 21.1 Platform Designer enforces this). With 0, the slave's
# waitrequest is the sole write back-pressure mechanism - same as the
```

```
# DE1-SoC DE1_SOC_Linux_FB demo's master_secure.
set_interface_property avalon_master_0 maximumPendingWriteTransactions 0
set_interface_property avalon_master_0 readLatency 0
set_interface_property avalon_master_0 readWaitTime 0
set_interface_property avalon_master_0 setupTime 0
set_interface_property avalon_master_0 timingUnits Cycles
set_interface_property avalon_master_0 writeWaitTime 0
set_interface_property avalon_master_0 ENABLED true
set_interface_property avalon_master_0 EXPORT_OF ""
set_interface_property avalon_master_0 PORT_NAME_MAP ""
set_interface_property avalon_master_0 CMSIS_SVD_VARIABLES ""
set_interface_property avalon_master_0 SVD_ADDRESS_GROUP ""

add_interface_port avalon_master_0 m_avm_address     address     Output 32
add_interface_port avalon_master_0 m_avm_byteenable  byteenable  Output 8
add_interface_port avalon_master_0 m_avm_write       write       Output 1
add_interface_port avalon_master_0 m_avm_writedata   writedata   Output 64
add_interface_port avalon_master_0 m_avm_burstcount  burstcount  Output 8
add_interface_port avalon_master_0 m_avm_waitrequest waitrequest Input  1
```

soc_system_assignment_defaults.qdf

```
set_global_assignment -name OPTIMIZATION_MODE "AGGRESSIVE PERFORMANCE"
set_global_assignment -name ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP ON
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION ON
set_global_assignment -name PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING ON
set_global_assignment -name SAVE_DISK_SPACE OFF
set_global_assignment -name TIMING_ANALYZER_MULTICORNER_ANALYSIS ON
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
set_global_assignment -name MUX_RESTRUCTURE ON
set_global_assignment -name REMOVE_REDUNDANT_LOGIC_CELLS ON
set_global_assignment -name AUTO_RESOURCE_SHARING ON
set_global_assignment -name OPTIMIZE_POWER_DURING_SYNTHESIS OFF
set_global_assignment -name SYNTH_PROTECT_SDC_CONSTRAINT ON
set_global_assignment -name PRE_MAPPING_RESYNTHESIS ON
set_global_assignment -name ROUTER_CLOCKING_TOPOLOGY_ANALYSIS ON
set_global_assignment -name OPTIMIZE_MULTI_CORNER_TIMING OFF
set_global_assignment -name OPTIMIZE_POWER_DURING_FITTING OFF
set_global_assignment -name PERIPHERY_TO_CORE_PLACEMENT_AND_ROUTING_OPTIMIZATION ON
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA ON
set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT EXTRA
set_global_assignment -name ROUTER_LCELL_INSERTION_AND_LOGIC_DUPLICATION ON
set_global_assignment -name ALM_REGISTER_PACKING_EFFORT HIGH

# Beneficial skew optimization globally (currently only enabled for SDRAM
# in soc_system.qsf). Quartus inserts deliberate clock-tree skew on
# destination FFs to borrow setup time from less-critical paths. The
# lx_r → light_x_local path already shows -0.478 ns clock skew helping;
# this option may push that further.
set_global_assignment -name ENABLE_BENEFICIAL_SKEW_OPTIMIZATION ON
set_global_assignment -name ENABLE_BENEFICIAL_SKEW_OPTIMIZATION_FOR_NON_GLOBAL_CLOCKS ON

set_global_assignment -name SEED 2

# Promote rtl_batch_pipe broadcast source FFs to global routing networks.
# Cyclone V has 16 dedicated global networks (CLKCTRL_G*) that can carry
# data with very low IC delay across the chip. The default routing for
# our 7.13 ns →FFFF broadcast IC may benefit from the dedicated tree.
# Caveat: global networks add ~1 ns clock-buffer overhead at the source,
# so net gain depends on Quartus's tradeoff. Failed pin assignments
# above are removed (rejected by Lite Edition).
set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|lx_r
set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|ly_r
set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|lz_r
```

soc_system.qsys

```
<?xml version="1.0" encoding="UTF-8"?>
<system name="$${FILENAME}">
 <component
```

```
   name="$${FILENAME}"
   displayName="$${FILENAME}"
   version="1.0"
   description=""
   tags=""
   categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
   element clk_0
   {
      datum _sortIndex
      {
         value = "0";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element hps_0
   {
      datum _sortIndex
      {
         value = "1";
         type = "int";
      }
   }
   element pll_0
   {
      datum _sortIndex
      {
         value = "3";
         type = "int";
      }
   }
   element raytracer_0
   {
      datum _sortIndex
      {
         value = "2";
         type = "int";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
         type = "String";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
         type = "String";
      }
   }
}
]]></parameter>
 <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="deviceFamily" value="Cyclone V" />
 <parameter name="deviceSpeedGrade" value="6" />
 <parameter name="fabricMode" value="QSYS" />
 <parameter name="generateLegacySim" value="false" />
 <parameter name="generationId" value="0" />
 <parameter name="globalResetBus" value="false" />
 <parameter name="hdlLanguage" value="VERILOG" />
 <parameter name="hideFromIPCatalog" value="false" />
 <parameter name="lockedInterfaceDefinition" value="" />
```

```
 <parameter name="maxAdditionalLatency" value="2" />
 <parameter name="projectName" value="soc_system.qpf" />
 <parameter name="sopcBorderPoints" value="false" />
 <parameter name="systemHash" value="0" />
 <parameter name="testBenchDutName" value="" />
 <parameter name="timeStamp" value="0" />
 <parameter name="useTestBenchNamingPattern" value="false" />
 <instanceScript></instanceScript>
 <interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
 <interface name="hps" internal="hps_0.hps_io" type="conduit" dir="end" />
 <interface name="hps_ddr3" internal="hps_0.memory" type="conduit" dir="end" />
 <interface name="pll_0_locked" internal="pll_0.locked" type="conduit" dir="end" />
 <interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
 <module name="clk_0" kind="clock_source" version="21.1" enabled="1">
  <parameter name="clockFrequency" value="100000000" />
  <parameter name="clockFrequencyKnown" value="true" />
  <parameter name="inputClockFrequency" value="0" />
  <parameter name="resetSynchronousEdges" value="NONE" />
 </module>
 <module name="hps_0" kind="altera_hps" version="21.1" enabled="1">
  <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
  <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
  <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
  <parameter name="AC_PACKAGE_DESKEW" value="false" />
  <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
  <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
  <parameter name="ADDR_ORDER" value="0" />
  <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
  <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
  <parameter name="ADVANCED_CK_PHASES" value="false" />
  <parameter name="ADVERTIZE_SEQUENCER_SW_BUILD_FILES" value="false" />
  <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
  <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
  <parameter name="AP_MODE" value="false" />
  <parameter name="AP_MODE_EN" value="0" />
  <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
  <parameter name="AUTO_PD_CYCLES" value="0" />
  <parameter name="AUTO_POWERDN_EN" value="false" />
  <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
  <parameter name="AVL_MAX_SIZE" value="4" />
  <parameter name="BONDING_OUT_ENABLED" value="false" />
  <parameter name="BOOTFROMFPGA_Enable" value="false" />
  <parameter name="BSEL" value="1" />
  <parameter name="BSEL_EN" value="false" />
  <parameter name="BYTE_ENABLE" value="true" />
  <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
  <parameter name="CALIBRATION_MODE" value="Skip" />
  <parameter name="CALIB_REG_WIDTH" value="8" />
  <parameter name="CAN0_Mode" value="N/A" />
  <parameter name="CAN0_PinMuxing" value="Unused" />
  <parameter name="CAN1_Mode" value="N/A" />
  <parameter name="CAN1_PinMuxing" value="Unused" />
  <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
  <parameter name="CFG_REORDER_DATA" value="true" />
  <parameter name="CFG_TCCD_NS" value="2.5" />
  <parameter name="COMMAND_PHASE" value="0.0" />
  <parameter name="CONTROLLER_LATENCY" value="5" />
  <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
  <parameter name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional,
Bidirectional</parameter>
  <parameter name="CSEL" value="0" />
  <parameter name="CSEL_EN" value="false" />
  <parameter name="CTI_Enable" value="false" />
  <parameter name="CTL_AUTOPCH_EN" value="false" />
  <parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
  <parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
  <parameter name="CTL_CSR_ENABLED" value="false" />
  <parameter name="CTL_CSR_READ_ONLY" value="1" />
  <parameter name="CTL_DEEP_POWERDN_EN" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
  <parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
  <parameter name="CTL_ECC_ENABLED" value="false" />
  <parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />
```

```
<parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
<parameter name="CTL_HRB_ENABLED" value="false" />
<parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
<parameter name="CTL_SELF_REFRESH_EN" value="false" />
<parameter name="CTL_USR_REFRESH_EN" value="false" />
<parameter name="CTL_ZQCAL_EN" value="false" />
<parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
<parameter name="DAT_DATA_WIDTH" value="32" />
<parameter name="DEBUGAPB_Enable" value="false" />
<parameter name="DEBUG_MODE" value="false" />
<parameter name="DEVICE_DEPTH" value="1" />
<parameter name="DEVICE_FAMILY_PARAM" value="" />
<parameter name="DISABLE_CHILD_MESSAGING" value="false" />
<parameter name="DISCRETE_FLY_BY" value="true" />
<parameter name="DLL_SHARING_MODE" value="None" />
<parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
<parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
<parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
<parameter name="DUPLICATE_AC" value="false" />
<parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
<parameter name="EMAC0_Mode" value="N/A" />
<parameter name="EMAC0_PTP" value="false" />
<parameter name="EMAC0_PinMuxing" value="Unused" />
<parameter name="EMAC1_Mode" value="RGMII" />
<parameter name="EMAC1_PTP" value="false" />
<parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
<parameter name="ENABLE_BONDING" value="false" />
<parameter name="ENABLE_BURST_MERGE" value="false" />
<parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
<parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
<parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
<parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
<parameter name="ENABLE_EXTRA_REPORTING" value="false" />
<parameter name="ENABLE_ISS_PROBES" value="false" />
<parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
<parameter name="ENABLE_NON_DES_CAL" value="false" />
<parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
<parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
<parameter name="ENABLE_USER_ECC" value="false" />
<parameter name="EXPORT_AFI_HALF_CLK" value="false" />
<parameter name="EXTRA_SETTINGS" value="" />
<parameter name="F2H_AXI_CLOCK_FREQ" value="100000000" />
<parameter name="F2H_SDRAM0_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
<parameter name="F2SCLK_COLDRST_Enable" value="false" />
<parameter name="F2SCLK_DBGRST_Enable" value="false" />
<parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
<parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
<parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
<parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
<parameter name="F2SCLK_WARMRST_Enable" value="false" />
<parameter name="F2SDRAM_Type" value="" />
<parameter name="F2SDRAM_Width" value="" />
<parameter name="F2SINTERRUPT_Enable" value="false" />
<parameter name="F2S_Width" value="2" />
<parameter name="FIX_READ_LATENCY" value="8" />
<parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
<parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
<parameter name="FORCE_DQS_TRACKING" value="AUTO" />
<parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
<parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
<parameter name="FORCE_SHADOW_REGS" value="AUTO" />
<parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
<parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN" value="100" />
<parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN" value="100" />
<parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN" value="100" />
<parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN" value="100" />
<parameter
   name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"
```

```xml
      value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK" value="125" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK" value="2.5" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK" value="125" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK" value="2.5" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT" value="100" />
  <parameter name="GPIO_Enable">No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,No,No,No,No,No,No,Yes,No,No,No,No,Yes,Yes,No,No,No,No,No,No,Yes,No,
No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No</
parameter>
  <parameter name="GP_Enable" value="false" />
  <parameter name="H2F_AXI_CLOCK_FREQ" value="100000000" />
  <parameter name="H2F_CTI_CLOCK_FREQ" value="100" />
  <parameter name="H2F_DEBUG_APB_CLOCK_FREQ" value="100" />
  <parameter name="H2F_LW_AXI_CLOCK_FREQ" value="100000000" />
  <parameter name="H2F_TPIU_CLOCK_IN_FREQ" value="100" />
  <parameter name="HARD_EMIF" value="true" />
  <parameter name="HCX_COMPAT_MODE" value="false" />
  <parameter name="HHP_HPS" value="true" />
  <parameter name="HHP_HPS_SIMULATION" value="false" />
  <parameter name="HHP_HPS_VERIFICATION" value="false" />
  <parameter name="HLGPI_Enable" value="false" />
  <parameter name="HPS_PROTOCOL" value="DDR3" />
  <parameter name="I2C0_Mode" value="I2C" />
  <parameter name="I2C0_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="I2C1_Mode" value="I2C" />
  <parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="I2C2_Mode" value="N/A" />
  <parameter name="I2C2_PinMuxing" value="Unused" />
  <parameter name="I2C3_Mode" value="N/A" />
  <parameter name="I2C3_PinMuxing" value="Unused" />
  <parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
  <parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
  <parameter name="IS_ES_DEVICE" value="false" />
  <parameter name="LOANIO_Enable">No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No</parameter
>
  <parameter name="LOCAL_ID_WIDTH" value="8" />
  <parameter name="LRDIMM_EXTENDED_CONFIG">0x000000000000000000</parameter>
  <parameter name="LWH2F_Enable" value="true" />
  <parameter name="MARGIN_VARIATION_TEST" value="false" />
  <parameter name="MAX_PENDING_RD_CMD" value="32" />
  <parameter name="MAX_PENDING_WR_CMD" value="16" />
  <parameter name="MEM_ASR" value="Manual" />
  <parameter name="MEM_ATCL" value="Disabled" />
  <parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
  <parameter name="MEM_BANKADDR_WIDTH" value="3" />
  <parameter name="MEM_BL" value="OTF" />
  <parameter name="MEM_BT" value="Sequential" />
  <parameter name="MEM_CK_PHASE" value="0.0" />
  <parameter name="MEM_CK_WIDTH" value="1" />
  <parameter name="MEM_CLK_EN_WIDTH" value="1" />
  <parameter name="MEM_CLK_FREQ" value="400.0" />
  <parameter name="MEM_CLK_FREQ_MAX" value="800.0" />
  <parameter name="MEM_COL_ADDR_WIDTH" value="10" />
  <parameter name="MEM_CS_WIDTH" value="1" />
  <parameter name="MEM_DEVICE" value="MISSING_MODEL" />
  <parameter name="MEM_DLL_EN" value="true" />
```

```
<parameter name="MEM_DQ_PER_DQS" value="8" />
<parameter name="MEM_DQ_WIDTH" value="32" />
<parameter name="MEM_DRV_STR" value="RZQ/7" />
<parameter name="MEM_FORMAT" value="DISCRETE" />
<parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
<parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
<parameter name="MEM_IF_DM_PINS_EN" value="true" />
<parameter name="MEM_IF_DQSN_EN" value="true" />
<parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
<parameter name="MEM_INIT_EN" value="false" />
<parameter name="MEM_INIT_FILE" value="" />
<parameter name="MEM_MIRROR_ADDRESSING" value="0" />
<parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
<parameter name="MEM_PD" value="DLL off" />
<parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
<parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
<parameter name="MEM_RTT_NOM" value="RZQ/4" />
<parameter name="MEM_RTT_WR" value="RZQ/4" />
<parameter name="MEM_SRT" value="Normal" />
<parameter name="MEM_TCL" value="11" />
<parameter name="MEM_TFAW_NS" value="30.0" />
<parameter name="MEM_TINIT_US" value="500" />
<parameter name="MEM_TMRD_CK" value="4" />
<parameter name="MEM_TRAS_NS" value="35.0" />
<parameter name="MEM_TRCD_NS" value="13.75" />
<parameter name="MEM_TREFI_US" value="7.8" />
<parameter name="MEM_TRFC_NS" value="260.0" />
<parameter name="MEM_TRP_NS" value="13.75" />
<parameter name="MEM_TRRD_NS" value="7.7" />
<parameter name="MEM_TRTP_NS" value="7.5" />
<parameter name="MEM_TWR_NS" value="15.0" />
<parameter name="MEM_TWTR" value="4" />
<parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
<parameter name="MEM_VENDOR" value="JEDEC" />
<parameter name="MEM_VERBOSE" value="true" />
<parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
<parameter name="MEM_WTCL" value="8" />
<parameter name="MPU_EVENTS_Enable" value="false" />
<parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
<parameter name="MULTICAST_EN" value="false" />
<parameter name="NAND_Mode" value="N/A" />
<parameter name="NAND_PinMuxing" value="Unused" />
<parameter name="NEXTGEN" value="true" />
<parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
<parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
<parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
<parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
<parameter name="NUM_OF_PORTS" value="1" />
<parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
<parameter name="OCT_SHARING_MODE" value="None" />
<parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
<parameter name="PACKAGE_DESKEW" value="false" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
<parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="PHY_CSR_ENABLED" value="false" />
<parameter name="PHY_ONLY" value="false" />
<parameter name="PINGPONGPHY_EN" value="false" />
<parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
```

```
<parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_CLK_PARAM_VALID" value="false" />
<parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_LOCATION" value="Top_Bottom" />
<parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_SHARING_MODE" value="None" />
<parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="POWER_OF_TWO_BUS" value="false" />
<parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
<parameter name="QSPI_Mode" value="N/A" />
<parameter name="QSPI_PinMuxing" value="Unused" />
<parameter name="RATE" value="Full" />
<parameter name="RDIMM_CONFIG" value="0000000000000000" />
<parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
<parameter name="READ_FIFO_SIZE" value="8" />
<parameter name="REFRESH_BURST_VALIDATION" value="false" />
<parameter name="REFRESH_INTERVAL" value="15000" />
<parameter name="REF_CLK_FREQ" value="25.0" />
<parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
```

```
<parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
<parameter name="S2FCLK_COLDRST_Enable" value="false" />
<parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
<parameter name="S2FCLK_USER0CLK_Enable" value="false" />
<parameter name="S2FCLK_USER1CLK_Enable" value="true" />
<parameter name="S2FCLK_USER1CLK_FREQ" value="100.0" />
<parameter name="S2FCLK_USER2CLK" value="5" />
<parameter name="S2FCLK_USER2CLK_Enable" value="false" />
<parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
<parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
<parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
<parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
<parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
<parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
<parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
<parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
<parameter name="S2FINTERRUPT_UART_Enable" value="false" />
<parameter name="S2FINTERRUPT_USB_Enable" value="false" />
<parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
<parameter name="S2F_Width" value="2" />
<parameter name="SDIO_Mode" value="4-bit Data" />
<parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SEQUENCER_TYPE" value="NIOS" />
<parameter name="SEQ_MODE" value="0" />
<parameter name="SKIP_MEM_INIT" value="true" />
<parameter name="SOPC_COMPAT_RESET" value="false" />
<parameter name="SPEED_GRADE" value="7" />
<parameter name="SPIM0_Mode" value="N/A" />
<parameter name="SPIM0_PinMuxing" value="Unused" />
<parameter name="SPIM1_Mode" value="Single Slave Select" />
<parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIS0_Mode" value="N/A" />
<parameter name="SPIS0_PinMuxing" value="Unused" />
<parameter name="SPIS1_Mode" value="N/A" />
<parameter name="SPIS1_PinMuxing" value="Unused" />
<parameter name="STARVE_LIMIT" value="10" />
<parameter name="STM_Enable" value="false" />
<parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
<parameter name="TEST_Enable" value="false" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
<parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
<parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
<parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
<parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
<parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
<parameter name="TIMING_BOARD_TDH" value="0.0" />
<parameter name="TIMING_BOARD_TDS" value="0.0" />
<parameter name="TIMING_BOARD_TIH" value="0.0" />
<parameter name="TIMING_BOARD_TIS" value="0.0" />
```

```
  <parameter name="TIMING_TDH" value="65" />
  <parameter name="TIMING_TDQSCK" value="255" />
  <parameter name="TIMING_TDQSCKDL" value="1200" />
  <parameter name="TIMING_TDQSCKDM" value="900" />
  <parameter name="TIMING_TDQSCKDS" value="450" />
  <parameter name="TIMING_TDQSH" value="0.35" />
  <parameter name="TIMING_TDQSQ" value="125" />
  <parameter name="TIMING_TDQSS" value="0.25" />
  <parameter name="TIMING_TDS" value="30" />
  <parameter name="TIMING_TDSH" value="0.2" />
  <parameter name="TIMING_TDSS" value="0.2" />
  <parameter name="TIMING_TIH" value="140" />
  <parameter name="TIMING_TIS" value="180" />
  <parameter name="TIMING_TQH" value="0.38" />
  <parameter name="TIMING_TQHS" value="300" />
  <parameter name="TIMING_TQSH" value="0.4" />
  <parameter name="TPIUFPGA_Enable" value="false" />
  <parameter name="TPIUFPGA_alt" value="false" />
  <parameter name="TRACE_Mode" value="N/A" />
  <parameter name="TRACE_PinMuxing" value="Unused" />
  <parameter name="TRACKING_ERROR_TEST" value="false" />
  <parameter name="TRACKING_WATCH_TEST" value="false" />
  <parameter name="TREFI" value="35100" />
  <parameter name="TRFC" value="350" />
  <parameter name="UART0_Mode" value="No Flow Control" />
  <parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="UART1_Mode" value="N/A" />
  <parameter name="UART1_PinMuxing" value="Unused" />
  <parameter name="USB0_Mode" value="N/A" />
  <parameter name="USB0_PinMuxing" value="Unused" />
  <parameter name="USB1_Mode" value="SDR" />
  <parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="USER_DEBUG_LEVEL" value="1" />
  <parameter name="USE_AXI_ADAPTOR" value="false" />
  <parameter name="USE_FAKE_PHY" value="false" />
  <parameter name="USE_MEM_CLK_FREQ" value="false" />
  <parameter name="USE_MM_ADAPTOR" value="true" />
  <parameter name="USE_SEQUENCER_BFM" value="false" />
  <parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
  <parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
  <parameter name="can0_clk_div" value="1" />
  <parameter name="can1_clk_div" value="1" />
  <parameter name="configure_advanced_parameters" value="false" />
  <parameter name="customize_device_pll_info" value="false" />
  <parameter name="dbctrl_stayosc1" value="true" />
  <parameter name="dbg_at_clk_div" value="0" />
  <parameter name="dbg_clk_div" value="1" />
  <parameter name="dbg_trace_clk_div" value="0" />
  <parameter name="desired_can0_clk_mhz" value="100.0" />
  <parameter name="desired_can1_clk_mhz" value="100.0" />
  <parameter name="desired_cfg_clk_mhz" value="97.368421" />
  <parameter name="desired_emac0_clk_mhz" value="250.0" />
  <parameter name="desired_emac1_clk_mhz" value="250.0" />
  <parameter name="desired_gpio_db_clk_hz" value="32000" />
  <parameter name="desired_l4_mp_clk_mhz" value="100.0" />
  <parameter name="desired_l4_sp_clk_mhz" value="100.0" />
  <parameter name="desired_mpu_clk_mhz" value="800.0" />
  <parameter name="desired_nand_clk_mhz" value="12.5" />
  <parameter name="desired_qspi_clk_mhz" value="400.0" />
  <parameter name="desired_sdmmc_clk_mhz" value="200.0" />
  <parameter name="desired_spi_m_clk_mhz" value="200.0" />
  <parameter name="desired_usb_mp_clk_mhz" value="200.0" />
  <parameter name="device_name" value="5CSEMA5F31C6" />
  <parameter name="device_pll_info_manual">{320000000 1600000000} {320000000 1000000000} {800000000 400000000
400000000}</parameter>
  <parameter name="eosc1_clk_mhz" value="25.0" />
  <parameter name="eosc2_clk_mhz" value="25.0" />
  <parameter name="gpio_db_clk_div" value="6249" />
  <parameter name="l3_mp_clk_div" value="1" />
  <parameter name="l3_sp_clk_div" value="1" />
  <parameter name="l4_mp_clk_div" value="1" />
  <parameter name="l4_mp_clk_source" value="1" />
  <parameter name="l4_sp_clk_div" value="1" />
  <parameter name="l4_sp_clk_source" value="1" />
```

```
 <parameter name="main_pll_c3" value="3" />
 <parameter name="main_pll_c4" value="3" />
 <parameter name="main_pll_c5" value="15" />
 <parameter name="main_pll_m" value="63" />
 <parameter name="main_pll_n" value="0" />
 <parameter name="nand_clk_source" value="2" />
 <parameter name="periph_pll_c0" value="3" />
 <parameter name="periph_pll_c1" value="3" />
 <parameter name="periph_pll_c2" value="1" />
 <parameter name="periph_pll_c3" value="19" />
 <parameter name="periph_pll_c4" value="4" />
 <parameter name="periph_pll_c5" value="9" />
 <parameter name="periph_pll_m" value="79" />
 <parameter name="periph_pll_n" value="1" />
 <parameter name="periph_pll_source" value="0" />
 <parameter name="qspi_clk_source" value="1" />
 <parameter name="quartus_ini_hps_emif_pll" value="false" />
 <parameter
    name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
    value="false" />
 <parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
 <parameter
    name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
    value="false" />
 <parameter
    name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
    value="false" />
 <parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
 <parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false" />
 <parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model" value="false" />
 <parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false" />
 <parameter name="sdmmc_clk_source" value="2" />
 <parameter name="show_advanced_parameters" value="false" />
 <parameter name="show_debug_info_as_warning_msg" value="false" />
 <parameter name="show_warning_as_error_msg" value="false" />
 <parameter name="spi_m_clk_div" value="0" />
 <parameter name="usb_mp_clk_div" value="0" />
 <parameter name="use_default_mpu_clk" value="true" />
</module>
<module name="pll_0" kind="altera_pll" version="21.1" enabled="1">
 <parameter name="debug_print_output" value="false" />
 <parameter name="debug_use_rbc_taf_method" value="false" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="device_family" value="Cyclone V" />
 <parameter name="gui_active_clk" value="false" />
 <parameter name="gui_actual_output_clock_frequency0" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency1" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency10" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency11" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency12" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency13" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency14" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency15" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency16" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency17" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency2" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency3" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency4" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency5" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency6" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency7" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency8" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency9" value="0 MHz" />
 <parameter name="gui_actual_phase_shift0" value="0" />
 <parameter name="gui_actual_phase_shift1" value="0" />
 <parameter name="gui_actual_phase_shift10" value="0" />
 <parameter name="gui_actual_phase_shift11" value="0" />
 <parameter name="gui_actual_phase_shift12" value="0" />
 <parameter name="gui_actual_phase_shift13" value="0" />
 <parameter name="gui_actual_phase_shift14" value="0" />
 <parameter name="gui_actual_phase_shift15" value="0" />
 <parameter name="gui_actual_phase_shift16" value="0" />
 <parameter name="gui_actual_phase_shift17" value="0" />
 <parameter name="gui_actual_phase_shift2" value="0" />
```

```
<parameter name="gui_actual_phase_shift3" value="0" />
<parameter name="gui_actual_phase_shift4" value="0" />
<parameter name="gui_actual_phase_shift5" value="0" />
<parameter name="gui_actual_phase_shift6" value="0" />
<parameter name="gui_actual_phase_shift7" value="0" />
<parameter name="gui_actual_phase_shift8" value="0" />
<parameter name="gui_actual_phase_shift9" value="0" />
<parameter name="gui_cascade_counter0" value="false" />
<parameter name="gui_cascade_counter1" value="false" />
<parameter name="gui_cascade_counter10" value="false" />
<parameter name="gui_cascade_counter11" value="false" />
<parameter name="gui_cascade_counter12" value="false" />
<parameter name="gui_cascade_counter13" value="false" />
<parameter name="gui_cascade_counter14" value="false" />
<parameter name="gui_cascade_counter15" value="false" />
<parameter name="gui_cascade_counter16" value="false" />
<parameter name="gui_cascade_counter17" value="false" />
<parameter name="gui_cascade_counter2" value="false" />
<parameter name="gui_cascade_counter3" value="false" />
<parameter name="gui_cascade_counter4" value="false" />
<parameter name="gui_cascade_counter5" value="false" />
<parameter name="gui_cascade_counter6" value="false" />
<parameter name="gui_cascade_counter7" value="false" />
<parameter name="gui_cascade_counter8" value="false" />
<parameter name="gui_cascade_counter9" value="false" />
<parameter name="gui_cascade_outclk_index" value="0" />
<parameter name="gui_channel_spacing" value="0.0" />
<parameter name="gui_clk_bad" value="false" />
<parameter name="gui_device_speed_grade" value="1" />
<parameter name="gui_divide_factor_c0" value="1" />
<parameter name="gui_divide_factor_c1" value="1" />
<parameter name="gui_divide_factor_c10" value="1" />
<parameter name="gui_divide_factor_c11" value="1" />
<parameter name="gui_divide_factor_c12" value="1" />
<parameter name="gui_divide_factor_c13" value="1" />
<parameter name="gui_divide_factor_c14" value="1" />
<parameter name="gui_divide_factor_c15" value="1" />
<parameter name="gui_divide_factor_c16" value="1" />
<parameter name="gui_divide_factor_c17" value="1" />
<parameter name="gui_divide_factor_c2" value="1" />
<parameter name="gui_divide_factor_c3" value="1" />
<parameter name="gui_divide_factor_c4" value="1" />
<parameter name="gui_divide_factor_c5" value="1" />
<parameter name="gui_divide_factor_c6" value="1" />
<parameter name="gui_divide_factor_c7" value="1" />
<parameter name="gui_divide_factor_c8" value="1" />
<parameter name="gui_divide_factor_c9" value="1" />
<parameter name="gui_divide_factor_n" value="1" />
<parameter name="gui_dps_cntr" value="C0" />
<parameter name="gui_dps_dir" value="Positive" />
<parameter name="gui_dps_num" value="1" />
<parameter name="gui_dsm_out_sel" value="1st_order" />
<parameter name="gui_duty_cycle0" value="50" />
<parameter name="gui_duty_cycle1" value="50" />
<parameter name="gui_duty_cycle10" value="50" />
<parameter name="gui_duty_cycle11" value="50" />
<parameter name="gui_duty_cycle12" value="50" />
<parameter name="gui_duty_cycle13" value="50" />
<parameter name="gui_duty_cycle14" value="50" />
<parameter name="gui_duty_cycle15" value="50" />
<parameter name="gui_duty_cycle16" value="50" />
<parameter name="gui_duty_cycle17" value="50" />
<parameter name="gui_duty_cycle2" value="50" />
<parameter name="gui_duty_cycle3" value="50" />
<parameter name="gui_duty_cycle4" value="50" />
<parameter name="gui_duty_cycle5" value="50" />
<parameter name="gui_duty_cycle6" value="50" />
<parameter name="gui_duty_cycle7" value="50" />
<parameter name="gui_duty_cycle8" value="50" />
<parameter name="gui_duty_cycle9" value="50" />
<parameter name="gui_en_adv_params" value="false" />
<parameter name="gui_en_dps_ports" value="false" />
<parameter name="gui_en_phout_ports" value="false" />
<parameter name="gui_en_reconf" value="false" />
```

```
<parameter name="gui_enable_cascade_in" value="false" />
<parameter name="gui_enable_cascade_out" value="false" />
<parameter name="gui_enable_mif_dps" value="false" />
<parameter name="gui_feedback_clock" value="Global Clock" />
<parameter name="gui_frac_multiply_factor" value="1" />
<parameter name="gui_fractional_cout" value="32" />
<parameter name="gui_mif_generate" value="false" />
<parameter name="gui_multiply_factor" value="1" />
<parameter name="gui_number_of_clocks" value="1" />
<parameter name="gui_operation_mode" value="direct" />
<parameter name="gui_output_clock_frequency0" value="125.0" />
<parameter name="gui_output_clock_frequency1" value="100.0" />
<parameter name="gui_output_clock_frequency10" value="100.0" />
<parameter name="gui_output_clock_frequency11" value="100.0" />
<parameter name="gui_output_clock_frequency12" value="100.0" />
<parameter name="gui_output_clock_frequency13" value="100.0" />
<parameter name="gui_output_clock_frequency14" value="100.0" />
<parameter name="gui_output_clock_frequency15" value="100.0" />
<parameter name="gui_output_clock_frequency16" value="100.0" />
<parameter name="gui_output_clock_frequency17" value="100.0" />
<parameter name="gui_output_clock_frequency2" value="100.0" />
<parameter name="gui_output_clock_frequency3" value="100.0" />
<parameter name="gui_output_clock_frequency4" value="100.0" />
<parameter name="gui_output_clock_frequency5" value="100.0" />
<parameter name="gui_output_clock_frequency6" value="100.0" />
<parameter name="gui_output_clock_frequency7" value="100.0" />
<parameter name="gui_output_clock_frequency8" value="100.0" />
<parameter name="gui_output_clock_frequency9" value="100.0" />
<parameter name="gui_phase_shift0" value="0" />
<parameter name="gui_phase_shift1" value="0" />
<parameter name="gui_phase_shift10" value="0" />
<parameter name="gui_phase_shift11" value="0" />
<parameter name="gui_phase_shift12" value="0" />
<parameter name="gui_phase_shift13" value="0" />
<parameter name="gui_phase_shift14" value="0" />
<parameter name="gui_phase_shift15" value="0" />
<parameter name="gui_phase_shift16" value="0" />
<parameter name="gui_phase_shift17" value="0" />
<parameter name="gui_phase_shift2" value="0" />
<parameter name="gui_phase_shift3" value="0" />
<parameter name="gui_phase_shift4" value="0" />
<parameter name="gui_phase_shift5" value="0" />
<parameter name="gui_phase_shift6" value="0" />
<parameter name="gui_phase_shift7" value="0" />
<parameter name="gui_phase_shift8" value="0" />
<parameter name="gui_phase_shift9" value="0" />
<parameter name="gui_phase_shift_deg0" value="0.0" />
<parameter name="gui_phase_shift_deg1" value="0.0" />
<parameter name="gui_phase_shift_deg10" value="0.0" />
<parameter name="gui_phase_shift_deg11" value="0.0" />
<parameter name="gui_phase_shift_deg12" value="0.0" />
<parameter name="gui_phase_shift_deg13" value="0.0" />
<parameter name="gui_phase_shift_deg14" value="0.0" />
<parameter name="gui_phase_shift_deg15" value="0.0" />
<parameter name="gui_phase_shift_deg16" value="0.0" />
<parameter name="gui_phase_shift_deg17" value="0.0" />
<parameter name="gui_phase_shift_deg2" value="0.0" />
<parameter name="gui_phase_shift_deg3" value="0.0" />
<parameter name="gui_phase_shift_deg4" value="0.0" />
<parameter name="gui_phase_shift_deg5" value="0.0" />
<parameter name="gui_phase_shift_deg6" value="0.0" />
<parameter name="gui_phase_shift_deg7" value="0.0" />
<parameter name="gui_phase_shift_deg8" value="0.0" />
<parameter name="gui_phase_shift_deg9" value="0.0" />
<parameter name="gui_phout_division" value="1" />
<parameter name="gui_pll_auto_reset" value="Off" />
<parameter name="gui_pll_bandwidth_preset" value="Auto" />
<parameter name="gui_pll_cascading_mode">Create an adjpllin signal to connect with an upstream PLL</parameter>
<parameter name="gui_pll_mode" value="Integer-N PLL" />
<parameter name="gui_ps_units0" value="ps" />
<parameter name="gui_ps_units1" value="ps" />
<parameter name="gui_ps_units10" value="ps" />
<parameter name="gui_ps_units11" value="ps" />
<parameter name="gui_ps_units12" value="ps" />
```

```
 <parameter name="gui_ps_units13" value="ps" />
 <parameter name="gui_ps_units14" value="ps" />
 <parameter name="gui_ps_units15" value="ps" />
 <parameter name="gui_ps_units16" value="ps" />
 <parameter name="gui_ps_units17" value="ps" />
 <parameter name="gui_ps_units2" value="ps" />
 <parameter name="gui_ps_units3" value="ps" />
 <parameter name="gui_ps_units4" value="ps" />
 <parameter name="gui_ps_units5" value="ps" />
 <parameter name="gui_ps_units6" value="ps" />
 <parameter name="gui_ps_units7" value="ps" />
 <parameter name="gui_ps_units8" value="ps" />
 <parameter name="gui_ps_units9" value="ps" />
 <parameter name="gui_refclk1_frequency" value="100.0" />
 <parameter name="gui_refclk_switch" value="false" />
 <parameter name="gui_reference_clock_frequency" value="50.0" />
 <parameter name="gui_switchover_delay" value="0" />
 <parameter name="gui_switchover_mode">Automatic Switchover</parameter>
 <parameter name="gui_use_locked" value="true" />
</module>
<module name="raytracer_0" kind="raytracer" version="1.0" enabled="1">
 <parameter name="FRAC_BITS" value="13" />
 <parameter name="HEIGHT" value="360" />
 <parameter name="N" value="480" />
 <parameter name="WIDTH" value="480" />
 <parameter name="WORD" value="27" />
</module>
<connection
  kind="avalon"
  version="21.1"
  start="hps_0.h2f_axi_master"
  end="raytracer_0.avalon_slave_0">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="21.1"
  start="raytracer_0.avalon_master_0"
  end="hps_0.f2h_axi_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="clock"
  version="21.1"
  start="clk_0.clk"
  end="hps_0.f2h_axi_clock" />
<connection
  kind="clock"
  version="21.1"
  start="clk_0.clk"
  end="hps_0.h2f_axi_clock" />
<connection
  kind="clock"
  version="21.1"
  start="clk_0.clk"
  end="hps_0.h2f_lw_axi_clock" />
<connection kind="clock" version="21.1" start="clk_0.clk" end="pll_0.refclk" />
<connection
  kind="clock"
  version="21.1"
  start="pll_0.outclk0"
  end="raytracer_0.clock" />
<connection
  kind="reset"
  version="21.1"
  start="clk_0.clk_reset"
  end="raytracer_0.reset" />
<connection kind="reset" version="21.1" start="clk_0.clk_reset" end="pll_0.reset" />
<interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
<interconnectRequirement for="$system" name="qsys_mm.enableEccProtection" value="FALSE" />
```

```
 <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave" value="FALSE" />
 <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="2" />
</system>
```

soc_system.tcl

```
# Generate Quartus project files for the DE1-SoC board
#
# Stephen A. Edwards, Columbia University

# Invoke as
#
# quartus_sh -t soc_system.tcl

set project "soc_system"

# Top-level SystemVerilog file should be <project>_top.sv, with Verilog module
# <project>_top in it

set systemVerilogSource "${project}_top.sv"
set qip "${project}/synthesis/${project}.qip"

project_new $project -overwrite

foreach {name value} {
    FAMILY "Cyclone V"
    DEVICE 5CSEMA5F31C6

    PROJECT_OUTPUT_DIRECTORY output_files

    CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"

    NUM_PARALLEL_PROCESSORS 4

} { set_global_assignment -name $name $value }

set_global_assignment -name TOP_LEVEL_ENTITY "${project}_top"

foreach filename $systemVerilogSource {
    set_global_assignment -name SYSTEMVERILOG_FILE $filename
}

foreach filename $qip {
    set_global_assignment -name QIP_FILE $filename
}

# FPGA pin assignments

foreach {pin port} {
    PIN_AJ4 ADC_CS_N
    PIN_AK4 ADC_DIN
    PIN_AK3 ADC_DOUT
    PIN_AK2 ADC_SCLK

    PIN_K7 AUD_ADCDAT
    PIN_K8 AUD_ADCLRCK
    PIN_H7 AUD_BCLK
    PIN_J7 AUD_DACDAT
    PIN_H8 AUD_DACLRCK
    PIN_G7 AUD_XCK

    PIN_AA16 CLOCK2_50
    PIN_Y26 CLOCK3_50
    PIN_K14 CLOCK4_50
    PIN_AF14 CLOCK_50

    PIN_AK14 DRAM_ADDR[0]
    PIN_AH14 DRAM_ADDR[1]
    PIN_AG15 DRAM_ADDR[2]
    PIN_AE14 DRAM_ADDR[3]
    PIN_AB15 DRAM_ADDR[4]
    PIN_AC14 DRAM_ADDR[5]
```

```
    PIN_AD14 DRAM_ADDR[6]
    PIN_AF15 DRAM_ADDR[7]
    PIN_AH15 DRAM_ADDR[8]
    PIN_AG13 DRAM_ADDR[9]
    PIN_AG12 DRAM_ADDR[10]
    PIN_AH13 DRAM_ADDR[11]
    PIN_AJ14 DRAM_ADDR[12]
    PIN_AF13 DRAM_BA[0]
    PIN_AJ12 DRAM_BA[1]
    PIN_AF11 DRAM_CAS_N
    PIN_AK13 DRAM_CKE
    PIN_AH12 DRAM_CLK
    PIN_AG11 DRAM_CS_N
    PIN_AK6 DRAM_DQ[0]
    PIN_AJ7 DRAM_DQ[1]
    PIN_AK7 DRAM_DQ[2]
    PIN_AK8 DRAM_DQ[3]
    PIN_AK9 DRAM_DQ[4]
    PIN_AG10 DRAM_DQ[5]
    PIN_AK11 DRAM_DQ[6]
    PIN_AJ11 DRAM_DQ[7]
    PIN_AH10 DRAM_DQ[8]
    PIN_AJ10 DRAM_DQ[9]
    PIN_AJ9 DRAM_DQ[10]
    PIN_AH9 DRAM_DQ[11]
    PIN_AH8 DRAM_DQ[12]
    PIN_AH7 DRAM_DQ[13]
    PIN_AJ6 DRAM_DQ[14]
    PIN_AJ5 DRAM_DQ[15]
    PIN_AB13 DRAM_LDQM
    PIN_AE13 DRAM_RAS_N
    PIN_AK12 DRAM_UDQM
    PIN_AA13 DRAM_WE_N

    PIN_AA12 FAN_CTRL

    PIN_J12 FPGA_I2C_SCLK
    PIN_K12 FPGA_I2C_SDAT

    PIN_AC18 GPIO_0[0]
    PIN_Y17  GPIO_0[1]
    PIN_AD17 GPIO_0[2]
    PIN_Y18  GPIO_0[3]
    PIN_AK16 GPIO_0[4]
    PIN_AK18 GPIO_0[5]
    PIN_AK19 GPIO_0[6]
    PIN_AJ19 GPIO_0[7]
    PIN_AJ17 GPIO_0[8]
    PIN_AJ16 GPIO_0[9]
    PIN_AH18 GPIO_0[10]
    PIN_AH17 GPIO_0[11]
    PIN_AG16 GPIO_0[12]
    PIN_AE16 GPIO_0[13]
    PIN_AF16 GPIO_0[14]
    PIN_AG17 GPIO_0[15]
    PIN_AA18 GPIO_0[16]
    PIN_AA19 GPIO_0[17]
    PIN_AE17 GPIO_0[18]
    PIN_AC20 GPIO_0[19]
    PIN_AH19 GPIO_0[20]
    PIN_AJ20 GPIO_0[21]
    PIN_AH20 GPIO_0[22]
    PIN_AK21 GPIO_0[23]
    PIN_AD19 GPIO_0[24]
    PIN_AD20 GPIO_0[25]
    PIN_AE18 GPIO_0[26]
    PIN_AE19 GPIO_0[27]
    PIN_AF20 GPIO_0[28]
    PIN_AF21 GPIO_0[29]
    PIN_AF19 GPIO_0[30]
    PIN_AG21 GPIO_0[31]
    PIN_AF18 GPIO_0[32]
    PIN_AG20 GPIO_0[33]
    PIN_AG18 GPIO_0[34]
```

```
    PIN_AJ21 GPIO_0[35]

    PIN_AB17 GPIO_1[0]
    PIN_AA21 GPIO_1[1]
    PIN_AB21 GPIO_1[2]
    PIN_AC23 GPIO_1[3]
    PIN_AD24 GPIO_1[4]
    PIN_AE23 GPIO_1[5]
    PIN_AE24 GPIO_1[6]
    PIN_AF25 GPIO_1[7]
    PIN_AF26 GPIO_1[8]
    PIN_AG25 GPIO_1[9]
    PIN_AG26 GPIO_1[10]
    PIN_AH24 GPIO_1[11]
    PIN_AH27 GPIO_1[12]
    PIN_AJ27 GPIO_1[13]
    PIN_AK29 GPIO_1[14]
    PIN_AK28 GPIO_1[15]
    PIN_AK27 GPIO_1[16]
    PIN_AJ26 GPIO_1[17]
    PIN_AK26 GPIO_1[18]
    PIN_AH25 GPIO_1[19]
    PIN_AJ25 GPIO_1[20]
    PIN_AJ24 GPIO_1[21]
    PIN_AK24 GPIO_1[22]
    PIN_AG23 GPIO_1[23]
    PIN_AK23 GPIO_1[24]
    PIN_AH23 GPIO_1[25]
    PIN_AK22 GPIO_1[26]
    PIN_AJ22 GPIO_1[27]
    PIN_AH22 GPIO_1[28]
    PIN_AG22 GPIO_1[29]
    PIN_AF24 GPIO_1[30]
    PIN_AF23 GPIO_1[31]
    PIN_AE22 GPIO_1[32]
    PIN_AD21 GPIO_1[33]
    PIN_AA20 GPIO_1[34]
    PIN_AC22 GPIO_1[35]

    PIN_AE26 HEX0[0]
    PIN_AE27 HEX0[1]
    PIN_AE28 HEX0[2]
    PIN_AG27 HEX0[3]
    PIN_AF28 HEX0[4]
    PIN_AG28 HEX0[5]
    PIN_AH28 HEX0[6]

    PIN_AJ29 HEX1[0]
    PIN_AH29 HEX1[1]
    PIN_AH30 HEX1[2]
    PIN_AG30 HEX1[3]
    PIN_AF29 HEX1[4]
    PIN_AF30 HEX1[5]
    PIN_AD27 HEX1[6]

    PIN_AB23 HEX2[0]
    PIN_AE29 HEX2[1]
    PIN_AD29 HEX2[2]
    PIN_AC28 HEX2[3]
    PIN_AD30 HEX2[4]
    PIN_AC29 HEX2[5]
    PIN_AC30 HEX2[6]

    PIN_AD26 HEX3[0]
    PIN_AC27 HEX3[1]
    PIN_AD25 HEX3[2]
    PIN_AC25 HEX3[3]
    PIN_AB28 HEX3[4]
    PIN_AB25 HEX3[5]
    PIN_AB22 HEX3[6]

    PIN_AA24 HEX4[0]
    PIN_Y23  HEX4[1]
    PIN_Y24  HEX4[2]
```

```
PIN_W22  HEX4[3]
PIN_W24  HEX4[4]
PIN_V23  HEX4[5]
PIN_W25  HEX4[6]

PIN_V25  HEX5[0]
PIN_AA28 HEX5[1]
PIN_Y27  HEX5[2]
PIN_AB27 HEX5[3]
PIN_AB26 HEX5[4]
PIN_AA26 HEX5[5]
PIN_AA25 HEX5[6]

PIN_AA30 IRDA_RXD
PIN_AB30 IRDA_TXD

PIN_AA14 KEY[0]
PIN_AA15 KEY[1]
PIN_W15  KEY[2]
PIN_Y16  KEY[3]

PIN_V16 LEDR[0]
PIN_W16 LEDR[1]
PIN_V17 LEDR[2]
PIN_V18 LEDR[3]
PIN_W17 LEDR[4]
PIN_W19 LEDR[5]
PIN_Y19 LEDR[6]
PIN_W20 LEDR[7]
PIN_W21 LEDR[8]
PIN_Y21 LEDR[9]

PIN_AD7 PS2_CLK
PIN_AD9 PS2_CLK2
PIN_AE7 PS2_DAT
PIN_AE9 PS2_DAT2

PIN_AB12 SW[0]
PIN_AC12 SW[1]
PIN_AF9  SW[2]
PIN_AF10 SW[3]
PIN_AD11 SW[4]
PIN_AD12 SW[5]
PIN_AE11 SW[6]
PIN_AC9  SW[7]
PIN_AD10 SW[8]
PIN_AE12 SW[9]

PIN_H15 TD_CLK27
PIN_D2  TD_DATA[0]
PIN_B1  TD_DATA[1]
PIN_E2  TD_DATA[2]
PIN_B2  TD_DATA[3]
PIN_D1  TD_DATA[4]
PIN_E1  TD_DATA[5]
PIN_C2  TD_DATA[6]
PIN_B3  TD_DATA[7]
PIN_A5  TD_HS
PIN_F6  TD_RESET_N
PIN_A3  TD_VS

PIN_A13 VGA_R[0]
PIN_C13 VGA_R[1]
PIN_E13 VGA_R[2]
PIN_B12 VGA_R[3]
PIN_C12 VGA_R[4]
PIN_D12 VGA_R[5]
PIN_E12 VGA_R[6]
PIN_F13 VGA_R[7]

PIN_J9  VGA_G[0]
PIN_J10 VGA_G[1]
PIN_H12 VGA_G[2]
PIN_G10 VGA_G[3]
```

```
    PIN_G11 VGA_G[4]
    PIN_G12 VGA_G[5]
    PIN_F11 VGA_G[6]
    PIN_E11 VGA_G[7]

    PIN_B13 VGA_B[0]
    PIN_G13 VGA_B[1]
    PIN_H13 VGA_B[2]
    PIN_F14 VGA_B[3]
    PIN_H14 VGA_B[4]
    PIN_F15 VGA_B[5]
    PIN_G15 VGA_B[6]
    PIN_J14 VGA_B[7]

    PIN_A11 VGA_CLK
    PIN_B11 VGA_HS
    PIN_D11 VGA_VS
    PIN_F10 VGA_BLANK_N
    PIN_C10 VGA_SYNC_N
} {
    set_location_assignment $pin -to $port
    set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
}

# HPS assignments

# 3.3-V LVTTL pins
foreach port {
    HPS_CONV_USB_N
    HPS_ENET_GTX_CLK
    HPS_ENET_INT_N
    HPS_ENET_MDC
    HPS_ENET_MDIO
    HPS_ENET_RX_CLK
    HPS_ENET_RX_DATA[0]
    HPS_ENET_RX_DATA[1]
    HPS_ENET_RX_DATA[2]
    HPS_ENET_RX_DATA[3]
    HPS_ENET_RX_DV
    HPS_ENET_TX_DATA[0]
    HPS_ENET_TX_DATA[1]
    HPS_ENET_TX_DATA[2]
    HPS_ENET_TX_DATA[3]
    HPS_ENET_TX_EN
    HPS_GSENSOR_INT
    HPS_I2C1_SCLK
    HPS_I2C1_SDAT
    HPS_I2C2_SCLK
    HPS_I2C2_SDAT
    HPS_I2C_CONTROL
    HPS_KEY
    HPS_LED
    HPS_LTC_GPIO
    HPS_SD_CLK
    HPS_SD_CMD
    HPS_SD_DATA[0]
    HPS_SD_DATA[1]
    HPS_SD_DATA[2]
    HPS_SD_DATA[3]
    HPS_SPIM_CLK
    HPS_SPIM_MISO
    HPS_SPIM_MOSI
    HPS_SPIM_SS
    HPS_UART_RX
    HPS_UART_TX
    HPS_USB_CLKOUT
    HPS_USB_DATA[0]
    HPS_USB_DATA[1]
    HPS_USB_DATA[2]
    HPS_USB_DATA[3]
    HPS_USB_DATA[4]
    HPS_USB_DATA[5]
    HPS_USB_DATA[6]
    HPS_USB_DATA[7]
```

```
    HPS_USB_DIR
    HPS_USB_NXT
    HPS_USB_STP
} {
    set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
}

# There are a lot of settings for the HPS_DDR3 interface not listed here.
# Instead, the
#
# soc_system/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
#
# script generated by qsys adds that information.  However, quartus_map
# must be run before this .tcl script may run because the script
# relies on being able to look at the (HPS) netlist to determine which
# pins to constrain

set sdcFilename "${project}.sdc"

set_global_assignment -name SDC_FILE $sdcFilename

set sdcf [open $sdcFilename "w"]
puts $sdcf {
    foreach {clock port} {
    clock_50_1 CLOCK_50
    clock_50_2 CLOCK2_50
    clock_50_3 CLOCK3_50
    clock_50_4 CLOCK4_50
    } {
    create_clock -name $clock -period 20ns [get_ports $port]
    }

    create_clock -name clock_27_1 -period 37 [get_ports TD_CLK27]

    derive_pll_clocks -create_base_clocks
    derive_clock_uncertainty
}
close $sdcf

project_close
```

2 Software (HPS Driver + Userspace Server)

SW/raytracer_batch.h

```
/* raytracer_batch.h */
#ifndef RAYTRACER_BATCH_H
#define RAYTRACER_BATCH_H

#include <linux/types.h>
#include <linux/ioctl.h>

/* Batch size (lanes per FPGA compute kick). Must equal the FPGA's
 * raytracer_0.N parameter - these are not validated against each other,
 * a mismatch silently corrupts data.
 *
 * To raise above 480, all of the following must be updated together:
 *
 *   THIS FILE:
 *     RTB_N (this define)
 *
 *   FPGA, parameter values:
 *     soc_system.qsys           <parameter name="N" value="..."/>
 *     raytracer_hw.tcl          add_parameter N INTEGER ... (default)
 *     raytracer.sv              parameter int N = ... (default)
 *
 *   FPGA, things that scale with N (currently sized for N  510):
 *     raytracer.sv              `8'd240` literal in DMA_IDLE init of
 *                               beats_remaining; should be (N/2)
 *     raytracer.sv              `8'd240` literal driving m_avm_burstcount;
 *                               same fix
```

```
 *     raytracer.sv              `beats_remaining` is 8 bits - widen to
 *                               9+ bits if N/2 > 255 (i.e. N  512)
 *     raytracer_hw.tcl          m_avm_burstcount port width is 8 - widen
 *                               to 9+ if N  512
 *
 *   FPGA, address span (currently sized for N+11  1024):
 *     The Avalon-MM slave's `address` is 10 bits = 1024 word-aperture.
 *     Control regs use 11 words (0..10), color mirror uses N words
 *     (11..N+10). At N=1013 this hits the limit.
 *     raytracer_hw.tcl          avalon_slave_0 address port width
 *     raytracer.sv              address [9:0] in module port list
 *
 *   AXI3 4 KB burst-boundary rule (per ARM IHI 0022):
 *     At N=1024 the 240-beat -> 4096-byte burst spans the full page.
 *     At N>1024 (>4096 B) the qsys bridge can't split a single Avalon
 *     burst across a 4 KB boundary; the master must issue >1 Avalon
 *     burst per batch. alloc_pages_exact always returns 4 KB-aligned,
 *     so the buffer base is fine - it's the in-burst byte count that
 *     matters.
 *
 *   render_frame.c:
 *     BATCHES = WIDTH / RTB_N. With WIDTH=480, BATCHES=1 at N=480 and
 *     0 at N>480 - the rendering loop wouldn't run. Going above 480
 *     means batches now span multiple rows; rework the pixel-address
 *     stride and the row-write logic.
 *
 *   Compute-side capacity:
 *     rtl_batch_pipe latency scales linearly with BATCH (~37*N/K
 *     cycles, K=3). At N=480 ~60 us; at N=960 ~120 us. Already faster
 *     than the DMA-out + commit at any N up through ~1000 lanes; if
 *     pushing further, may want K=4 to keep compute off the critical
 *     path.
 */
#define RTB_N 480

/* Block until the in-flight FRAME_DONE asserts, clear the sticky bit, and
 * invalidate the just-finished slot's CPU caches so userspace can read
 * cached. Returns the index (0 or 1) of the slot whose render just
 * committed; userspace reads at `frame + slot * (WIDTH*HEIGHT)`.
 *
 * Replaces the userspace `while (csr->status & FRAME_DONE) ;` poll +
 * `csr->control = CLEAR_FRAME_DONE` write that the mmap-based path
 * used to do inline. The reason this is a kernel ioctl (rather than
 * pure userspace polling) is the cache invalidate: the frame buffer
 * is now allocated cached for ~6× faster CPU reads, and on Cortex-A9
 * + PL310 the L2 invalidate after DMA-write is privileged. The poll
 * + clear are folded in to keep userspace at one syscall per frame. */
#define RTB_IOCTL_WAIT_FRAME _IOR('r', 3, __u32)

/* ============================================================
 * mmap interface (Stage 4 - HW per-frame auto-kick + frame-level double
 * buffering with pipelined render).
 * ============================================================
 *
 * Userspace mmaps two regions, each one page-aligned offset:
 *
 *   mmap(fd, off=RTB_MMAP_CSR_OFFSET)    → CSR page, RW, uncached.
 *                                          Cast to volatile rtb_csrs *.
 *   mmap(fd, off=RTB_MMAP_FRAME_OFFSET)  → 2-slot frame ring, R, cached
 *                                          (write-back). Each slot holds
 *                                          one full WIDTH×HEIGHT×4 frame
 *                                          in packed {00, R, G, B} per
 *                                          pixel, indexed (y*WIDTH+x).
 *                                          Total mapping is 2 * frame size.
 *                                          Coherence with FPGA writes is
 *                                          maintained by RTB_IOCTL_WAIT_FRAME
 *                                          which invalidates the just-
 *                                          finished slot before returning.
 *
 * Stage 4 double-buffer slot indexing:
 *   - HW maintains an internal write_slot bit (1 = slot 1, 0 = slot 0)
 *     that toggles after each FRAME_DONE_PULSE. Frame N lands in slot
 *     N%2.
 *   - SW receives the just-finished slot index from RTB_IOCTL_WAIT_FRAME
```

```
 *     and reads `frame + slot * (WIDTH * HEIGHT)` (or, equivalently,
 *     byte offset = slot * RTB_FRAME_BYTES_PER_SLOT).
 *
 * Pipelined wire protocol (server.c <-> client.py): pipeline-depth-2.
 *   - Client primes: send light 0 + light 1 immediately on connect.
 *   - Client steady: after recv frame N, send light N+2.
 *   - Server kicks frame N+1 BEFORE reading slot N, so HW renders
 *     N+1 in parallel with SW reading + sending N. Total per-frame
 *     wall clock = max(HW_render, SW_read_send)   HW_render.
 *
 * Driver programs FRAME_BUF_BASE (= base of slot 0) and DMA_CTRL[0]
 * at probe; userspace must NOT write those CSR fields. The struct
 * exposes frame_buf_base for completeness/debug only.
 */

#define RTB_MMAP_PAGE_SIZE     4096u
#define RTB_MMAP_CSR_OFFSET    (0u * RTB_MMAP_PAGE_SIZE)
#define RTB_MMAP_FRAME_OFFSET  (1u * RTB_MMAP_PAGE_SIZE)

/* Stage 4: bytes per slot (one full frame). The mmap'd frame region is
 * 2 * RTB_FRAME_BYTES_PER_SLOT total; SW indexes into the active slot. */
#define RTB_FRAME_BYTES_PER_SLOT (480u * 360u * 4u)

/* CSR layout - byte-exact mirror of the Avalon-MM slave register map
 * declared in raytracer.sv (localparam ADDR_* block). Cast a CSR mmap
 * to `volatile struct rtb_csrs *` and access fields directly.
 *
 * Per-frame loop: WAIT_FRAME → write all scene CSRs (light, spheres,
 * camera basis, visual params) → FRAME_START. SW must finish all writes
 * before kicking and never write CSRs while a frame is in flight; HW does
 * not double-buffer scene state.
 *
 * frame_buf_base is the DDR3 physical address of the 2-slot frame ring
 * (Stage 4): a 2*WIDTH*HEIGHT*4 contiguous DDR3 region the FPGA scatters
 * batch DMAs into. HW alternates which slot it writes on each
 * FRAME_DONE_PULSE. Driver programs it at probe; SW must NOT write
 * this field.
 *
 * Camera basis (right.y hardcoded 0 - no roll). SW computes from yaw +
 * pitch and ships these once per frame. HW applies
 * raw_dir = right·px + up·py + fwd in stage A. cam_y locked at 1.5.
 *
 * max_depth caps recursion (0..7; 0 disables reflection). reflect_shift_{0,1}
 * set per-sphere power-of-2 reflectivity (each bounce off sphere X
 * attenuates by >> reflect_shift_X). Defaults: depth=3, blue=25% (RS=2),
 * red=50% (RS=1). Cold-start defaults reproduce the historical hardcoded
 * scene exactly: cam at (0, 1.5, -8), identity basis (right=x̂, upŷ=, fwdẑ=).
 *
 * Q-format note: all signed-Q fields use FRAC_BITS = 13 (Q14.13). To
 * convert a float to wire format: int32_t v = (int32_t)(f * (1 << 13));
 * Negative values use two's-complement; HW takes writedata[WORD-1:0] so
 * masking the upper bits is unnecessary.
 */
struct rtb_csrs {
    __u32 control;            /* 0x00 W: bit2 frame_start, bit3 clear_frame_done */
    __u32 status;             /* 0x04 R: bit2 frame_done_sticky,
                               *        bit3 frame_running, bit4 write_slot */
    __u32 _rsvd_pixel_x;      /* 0x08 reserved (HW frame-mode generator owns the
                               *      internal pixel_x_reg; SW write path removed) */
    __u32 _rsvd_pixel_y;      /* 0x0c reserved (same as 0x08) */
    __u32 light_x;            /* 0x10 W: signed Q12, low 26 bits */
    __u32 light_y;            /* 0x14 W */
    __u32 light_z;            /* 0x18 W */
    __u32 r0_sq;              /* 0x1c W: signed Q FRAC_BITS sphere-0 radius²
                               *      (SW pre-squares r before writing). */
    __u32 r1_sq;              /* 0x20 W: signed Q FRAC_BITS sphere-1 radius² */
    __u32 dma_ctrl;           /* 0x24 W: bit0 enable - DRIVER-OWNED */
    __u32 dma_status;         /* 0x28 R: bit0 busy, bit1 err, bit2 done */
    __u32 _rsvd_2c;           /* 0x2c reserved (formerly LIGHT_COMMIT - collapsed
                               *      with NEXT/ACTIVE shadow regs) */
    __u32 _rsvd_30_38[3];     /* 0x30/0x34/0x38 reserved (former LIGHT_*_ACTIVE
                               *                debug readback) */
    __u32 frame_buf_base;     /* 0x3c W: DDR3 phys addr - DRIVER-OWNED */
    __u32 sc1_x;              /* 0x40 W: signed Q12 sphere-1 center X */
```

```
    __u32 sc1_y;              /* 0x44 W                              Y */
    __u32 sc1_z;              /* 0x48 W                              Z */
    __u32 max_depth;          /* 0x4c W: 3-bit max recursion depth (default 3) */
    __u32 reflect_shift_0;    /* 0x50 W: 5-bit pow2 reflectivity, sphere 0 (default 2 = 25%) */
    __u32 reflect_shift_1;    /* 0x54 W: 5-bit pow2 reflectivity, sphere 1 (default 1 = 50%) */
    /* Camera position (cam_y is locked in HW at 1.5 - no CSR). */
    __u32 cam_x;              /* 0x58 W: signed Q12 camera origin X (default 0.0) */
    __u32 cam_z;              /* 0x5c W: signed Q12 camera origin Z (default -8.0) */
    /* Camera basis (right.y hardcoded 0 - no roll). */
    __u32 right_x;            /* 0x60 W: signed Q12 right.x (default -1.0) */
    __u32 right_z;            /* 0x64 W: signed Q12 right.z (default  0.0) */
    __u32 up_x;               /* 0x68 W: signed Q12 up.x    (default  0.0) */
    __u32 up_y;               /* 0x6c W: signed Q12 up.y    (default  1.0) */
    __u32 up_z;               /* 0x70 W: signed Q12 up.z    (default  0.0) */
    __u32 fwd_x;              /* 0x74 W: signed Q12 fwd.x   (default  0.0) */
    __u32 fwd_y;              /* 0x78 W: signed Q12 fwd.y   (default  0.0) */
    __u32 fwd_z;              /* 0x7c W: signed Q12 fwd.z   (default  1.0) */
    /* Visual-upgrades CSRs. Defaults reproduce the legacy hardcoded scene
     * byte-equal so SW that doesn't write any of these still gets the old image. */
    __u32 sc0_x;              /* 0x80 W: signed Q12 sphere-0 center X (default 0.0) */
    __u32 sc0_y;              /* 0x84 W                              Y (default 1.0) */
    __u32 sc0_z;              /* 0x88 W                              Z (default 0.0) */
    __u32 pc1_r;              /* 0x8c W: signed Q12 floor color #1 R  (default 0.9) */
    __u32 pc1_g;              /* 0x90 W                            G  (default 0.9) */
    __u32 pc1_b;              /* 0x94 W                            B  (default 0.9) */
    __u32 pc2_r;              /* 0x98 W: signed Q12 floor color #2 R  (default 0.4) */
    __u32 pc2_g;              /* 0x9c W                            G  (default 0.4) */
    __u32 pc2_b;              /* 0xa0 W                            B  (default 0.4) */
    __u32 scol0_r;            /* 0xa4 W: signed Q12 sphere-0 color R  (default 0.2) */
    __u32 scol0_g;            /* 0xa8 W                            G  (default 0.3) */
    __u32 scol0_b;            /* 0xac W                            B  (default 0.8) */
    __u32 scol1_r;            /* 0xb0 W: signed Q12 sphere-1 color R  (default 0.8) */
    __u32 scol1_g;            /* 0xb4 W                            G  (default 0.3) */
    __u32 scol1_b;            /* 0xb8 W                            B  (default 0.2) */
    __u32 floor_mode;         /* 0xbc W: 2-bit floor pattern selector
                               *  00 = checker (default), 01 = Sierpinski,
                               *  10 = stripes, 11 = fine checker */
    __u32 floor_lim_x;        /* 0xc0 W: signed Q12 floor X half-extent
                               *         (default = max-positive  infinite) */
    __u32 floor_lim_z;        /* 0xc4 W: signed Q12 floor Z half-extent */
    /* Cone primitive (Y-axis cone, apex on top, opens downward over
     * y  [apex.y - height, apex.y]). Surface
     *   (x - Ax)² + (z - Az)² = k_sq · (y - Ay)²
     * Default height=0 disables the cone (slab is empty). reflect_shift
     * mirrors the per-sphere REFLECT_SHIFT_* CSRs. Sun-billboard uses
     * light/|light| automatically - no separate sun CSRs. */
    __u32 cone_x;             /* 0xc8 W: signed Q12 cone apex X (default 0) */
    __u32 cone_y;             /* 0xcc W: signed Q12 cone apex Y (default 0) */
    __u32 cone_z;             /* 0xd0 W: signed Q12 cone apex Z (default 0) */
    __u32 cone_k_sq;          /* 0xd4 W: signed Q12 tan²(half-angle) (default 1.0 = 45°) */
    __u32 cone_height;        /* 0xd8 W: signed Q12 cone height (default 0 = disabled) */
    __u32 cone_col_r;         /* 0xdc W: signed Q12 cone color R (default 0.5) */
    __u32 cone_col_g;         /* 0xe0 W                         G (default 0.5) */
    __u32 cone_col_b;         /* 0xe4 W                         B (default 0.5) */
    __u32 reflect_shift_cone; /* 0xe8 W: 5-bit pow2 reflectivity, cone (default 0 = matte) */
    __u32 cone_norm_factor;   /* 0xec W: signed Q12, sqrt(k²·(1+k²)). SW must
                               *         pre-compute this from cone_k_sq -
                               *         lets HW recover the cone surface
                               *         |normal| with one mul instead of
                               *         three squarings + a sqrt. Default
                               *         5793  √2·2¹² (= sqrt(2) for k²=1). */
    __u32 _rsvd_f0_fc[4];     /* 0xf0..0xfc reserved (padding to align color
                               *           mirror to word 64 / byte 0x100) */
    __u32 color[];            /* 0x100+ R: BRAM mirror, debug only. */
};

/* Bit defines for the polled fields. */
#define RTB_CONTROL_FRAME_START        0x00000004u
#define RTB_CONTROL_CLEAR_FRAME_DONE   0x00000008u
#define RTB_STATUS_FRAME_DONE          0x00000004u
#define RTB_STATUS_FRAME_RUNNING       0x00000008u
#define RTB_STATUS_WRITE_SLOT          0x00000010u   /* Stage 4: current write slot */

/* mmap layout returned by RTB_IOCTL_GET_LAYOUT. One-time call at
```

```
 * userspace setup; all values are stable for the device's lifetime. */
struct rtb_layout {
    __u32 csr_size;          /* size of CSR mmap (currently 4 KB) */
    __u32 frame_buf_bytes;   /* size of the frame buffer mmap (WIDTH*HEIGHT*4) */
    __u32 csr_offset;        /* RTB_MMAP_CSR_OFFSET (echoed for sanity) */
    __u32 frame_buf_offset;  /* RTB_MMAP_FRAME_OFFSET */
};

#define RTB_IOCTL_GET_LAYOUT _IOR('r', 2, struct rtb_layout)

#endif
```

SW/raytrace_batch_drve.c

```
/* raytracer_batch_drv.c */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/delay.h>
#include <linux/jiffies.h>
#include <linux/ktime.h>
#include <linux/slab.h>
#include <linux/dma-mapping.h>

#include "raytracer_batch.h"

#define DRIVER_NAME "raytracer"

/* CSR offsets the driver itself accesses. The full register layout lives in
 * `struct rtb_csrs` (raytracer_batch.h); userspace pokes everything else via
 * the CSR mmap. */
#define REG_CONTROL         0x00
#define REG_STATUS          0x04
#define REG_DMA_CTRL        0x24    /* bit0 ENABLE, bit1 CLEAR_ERR, bit2 CLEAR_DONE */
#define REG_FRAME_BUF_BASE  0x3c    /* W: DDR3 phys addr of contiguous frame buffer */

#define CONTROL_FRAME_START        0x00000004u
#define CONTROL_CLEAR_FRAME_DONE   0x00000008u

#define STATUS_FRAME_DONE          0x00000004u
#define STATUS_FRAME_RUNNING       0x00000008u

#define DMA_CTRL_ENABLE     0x00000001u
#define DMA_CTRL_CLEAR_ERR  0x00000002u
#define DMA_CTRL_CLEAR_DONE 0x00000004u

#define RTB_WIDTH      480u
#define RTB_HEIGHT     360u
#define RTB_TIMEOUT_MS 1000u

#define RTB_FRAME_BYTES_PER_SLOT  (RTB_WIDTH * RTB_HEIGHT * 4u)
#define RTB_FRAME_BYTES           (2u * RTB_FRAME_BYTES_PER_SLOT)

struct raytracer_batch_dev {
    struct resource res;
    void __iomem *virtbase;
    struct device *dma_dev;        /* parent device for streaming-DMA APIs */
    void          *cpu_frame;      /* cached kernel VA (alloc_pages_exact) */
    dma_addr_t     dma_frame;      /* bus addr sent to FPGA via FRAME_BUF_BASE */
} device_state;


/* RTB_IOCTL_WAIT_FRAME - block until FRAME_DONE, clear the sticky bit,
```

```
 * invalidate the just-finished slot's CPU caches, and return its slot
 * index (0 or 1). The L2 invalidate on Cortex-A9 + PL310 is privileged,
 * which is why poll + clear + invalidate live in one ioctl instead of
 * in userspace. */
static long raytracer_batch_ioctl_wait_frame(unsigned long arg)
{
    unsigned long deadline = jiffies + msecs_to_jiffies(RTB_TIMEOUT_MS);
    u32 status;
    __u32 done_slot;

    while (1) {
        status = ioread32(device_state.virtbase + REG_STATUS);
        if (status & STATUS_FRAME_DONE)
            break;
        if (time_after_eq(jiffies, deadline))
            return -ETIMEDOUT;
    }

    done_slot = (status & RTB_STATUS_WRITE_SLOT) ? 0u : 1u;

    iowrite32(CONTROL_CLEAR_FRAME_DONE,
          device_state.virtbase + REG_CONTROL);

    dma_sync_single_for_cpu(device_state.dma_dev,
                device_state.dma_frame
                    + done_slot * RTB_FRAME_BYTES_PER_SLOT,
                RTB_FRAME_BYTES_PER_SLOT,
                DMA_FROM_DEVICE);

    if (copy_to_user((void __user *)arg, &done_slot, sizeof(done_slot)))
        return -EFAULT;
    return 0;
}

/* RTB_IOCTL_GET_LAYOUT - one-time mmap-region descriptor. */
static long raytracer_batch_ioctl_get_layout(unsigned long arg)
{
    struct rtb_layout layout;

    memset(&layout, 0, sizeof(layout));
    layout.csr_size         = RTB_MMAP_PAGE_SIZE;
    layout.frame_buf_bytes  = RTB_FRAME_BYTES;
    layout.csr_offset       = RTB_MMAP_CSR_OFFSET;
    layout.frame_buf_offset = RTB_MMAP_FRAME_OFFSET;

    if (copy_to_user((void __user *)arg, &layout, sizeof(layout)))
        return -EFAULT;
    return 0;
}

static long raytracer_batch_ioctl(struct file *file, unsigned int cmd,
                  unsigned long arg)
{
    if (cmd == RTB_IOCTL_GET_LAYOUT)
        return raytracer_batch_ioctl_get_layout(arg);

    if (cmd == RTB_IOCTL_WAIT_FRAME)
        return raytracer_batch_ioctl_wait_frame(arg);

    return -EINVAL;
}

/* mmap handler - vm_pgoff selects which region:
 *   pgoff 0 → CSR window (uncached, RW). CSR side effects require no caching.
 *   pgoff 1 → frame buffer (cached, R). Coherence with FPGA DMA writes is
 *             maintained by RTB_IOCTL_WAIT_FRAME's per-slot invalidate. */
static int raytracer_batch_mmap(struct file *file, struct vm_area_struct *vma)
{
    unsigned long pgoff = vma->vm_pgoff;
    size_t size = vma->vm_end - vma->vm_start;
    int ret;

    switch (pgoff) {
    case 0: {
```

```
        size_t csr_phys_size = resource_size(&device_state.res);

        if (size > csr_phys_size)
            return -EINVAL;

        vma->vm_pgoff = 0;
        vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);
        ret = io_remap_pfn_range(vma, vma->vm_start,
                    device_state.res.start >> PAGE_SHIFT,
                    size, vma->vm_page_prot);
        return ret;
    }
    case 1: {
        unsigned long pfn;

        if (size > PAGE_ALIGN(RTB_FRAME_BYTES))
            return -EINVAL;

        pfn = virt_to_phys(device_state.cpu_frame) >> PAGE_SHIFT;
        ret = remap_pfn_range(vma, vma->vm_start, pfn, size,
                      vma->vm_page_prot);
        return ret;
    }
    default:
        return -EINVAL;
    }
}

static const struct file_operations raytracer_batch_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = raytracer_batch_ioctl,
    .mmap = raytracer_batch_mmap,
};

static struct miscdevice raytracer_batch_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DRIVER_NAME,
    .fops = &raytracer_batch_fops,
};

static int __init raytracer_batch_probe(struct platform_device *pdev)
{
    int ret_value;

    ret_value = misc_register(&raytracer_batch_misc_device);
    if (ret_value)
        return ret_value;

    ret_value = of_address_to_resource(pdev->dev.of_node, 0, &device_state.res);
    if (ret_value) {
        ret_value = -ENOENT;
        goto deregister_out;
    }

    if (!request_mem_region(device_state.res.start,
                resource_size(&device_state.res),
                DRIVER_NAME)) {
        ret_value = -EBUSY;
        goto deregister_out;
    }

    device_state.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (!device_state.virtbase) {
        ret_value = -ENOMEM;
        goto release_out;
    }

    device_state.dma_dev = &pdev->dev;
    ret_value = dma_set_mask_and_coherent(device_state.dma_dev,
                          DMA_BIT_MASK(32));
    if (ret_value)
        goto unmap_out;

    device_state.cpu_frame = alloc_pages_exact(RTB_FRAME_BYTES, GFP_KERNEL);
```

```
    if (!device_state.cpu_frame) {
        ret_value = -ENOMEM;
        goto unmap_out;
    }

    device_state.dma_frame = dma_map_single(device_state.dma_dev,
                        device_state.cpu_frame,
                        RTB_FRAME_BYTES,
                        DMA_FROM_DEVICE);
    if (dma_mapping_error(device_state.dma_dev, device_state.dma_frame)) {
        ret_value = -ENOMEM;
        goto free_frame_out;
    }

    if ((u64)device_state.dma_frame >= 0x80000000ull) {
        pr_err(DRIVER_NAME ": DMA addr unsuitable frame=%pad\n",
               &device_state.dma_frame);
        ret_value = -EINVAL;
        goto unmap_frame_out;
    }

    iowrite32((u32)device_state.dma_frame,
          device_state.virtbase + REG_FRAME_BUF_BASE);
    iowrite32(DMA_CTRL_CLEAR_DONE | DMA_CTRL_CLEAR_ERR,
          device_state.virtbase + REG_DMA_CTRL);
    iowrite32(DMA_CTRL_ENABLE,
          device_state.virtbase + REG_DMA_CTRL);

    pr_info(DRIVER_NAME ": frame buffer %pad (%u B)\n",
        &device_state.dma_frame, RTB_FRAME_BYTES);

    return 0;

unmap_frame_out:
    dma_unmap_single(device_state.dma_dev, device_state.dma_frame,
             RTB_FRAME_BYTES, DMA_FROM_DEVICE);
free_frame_out:
    free_pages_exact(device_state.cpu_frame, RTB_FRAME_BYTES);
unmap_out:
    iounmap(device_state.virtbase);
release_out:
    release_mem_region(device_state.res.start,
                resource_size(&device_state.res));
deregister_out:
    misc_deregister(&raytracer_batch_misc_device);
    return ret_value;
}

static int raytracer_batch_remove(struct platform_device *pdev)
{
    iowrite32(0u, device_state.virtbase + REG_DMA_CTRL);
    dma_unmap_single(device_state.dma_dev, device_state.dma_frame,
             RTB_FRAME_BYTES, DMA_FROM_DEVICE);
    free_pages_exact(device_state.cpu_frame, RTB_FRAME_BYTES);
    iounmap(device_state.virtbase);
    release_mem_region(device_state.res.start,
                resource_size(&device_state.res));
    misc_deregister(&raytracer_batch_misc_device);
    return 0;
}

#ifdef CONFIG_OF
static const struct of_device_id raytracer_batch_of_match[] = {
    { .compatible = "csee4840,raytracer-1.0" },
    { },
};
MODULE_DEVICE_TABLE(of, raytracer_batch_of_match);
#endif

static struct platform_driver raytracer_batch_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(raytracer_batch_of_match),
```

```
    },
    .remove = __exit_p(raytracer_batch_remove),
};

static int __init raytracer_batch_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&raytracer_batch_driver,
                  raytracer_batch_probe);
}

static void __exit raytracer_batch_exit(void)
{
    platform_driver_unregister(&raytracer_batch_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(raytracer_batch_init);
module_exit(raytracer_batch_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("raytracer batch driver");
```

SW/fpga/server.c

```
/*
 * server.c - TCP bridge for the FPGA raytracer batch device.
 *
 * Wire protocol (pipeline-depth-2):
 *   Client → server on connect: inputs 0, inputs 1 (priming).
 *   Steady state per frame:
 *     Server → client: full frame N (691,200 B BGRX, raw HW layout).
 *     Client → server: inputs N+2  (49 × i32 network-order; see
 *                                   client.pack_inputs for layout).
 *
 * Per frame: kick frame N+1 BEFORE sending N, so HW renders N+1 in
 * parallel with the SW send of N. Total wall time  max(HW, SW).
 *
 * On client disconnect, drain the in-flight frame before tearing
 * down the mmap so DMA doesn't write to freed user memory.
 *
 * Usage: ./server [-v] [port]      (default port 12345; -v = per-frame log)
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <signal.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>

#include "../raytracer_batch.h"

#define WIDTH       480
#define HEIGHT      360
#define FRAME_BYTES (WIDTH * HEIGHT * 4)    /* 691,200 - BGRX wire size */
#define SLOT_WORDS  (WIDTH * HEIGHT)        /* uint32_t per slot */
#define SLOT_BYTES  (SLOT_WORDS * 4u)       /* 691,200 - DDR slot stride */

#define DEFAULT_PORT 12345

static volatile sig_atomic_t stop;
static int verbose = 0;
```

```

static void sigint_handler(int sig)
{
    (void)sig;
    stop = 1;
}

static int recv_exact(int fd, void *buf, size_t n)
{
    size_t got = 0;
    while (got < n) {
        ssize_t r = recv(fd, (char *)buf + got, n - got, 0);
        if (r <= 0)
            return -1;
        got += (size_t)r;
    }
    return 0;
}

static int send_all(int fd, const void *buf, size_t n)
{
    size_t sent = 0;
    while (sent < n) {
        ssize_t w = send(fd, (const char *)buf + sent, n - sent, 0);
        if (w < 0)
            return -1;
        sent += (size_t)w;
    }
    return 0;
}

/* Per-frame inputs received from the client. cam_y is locked at 1.5 in HW
 * and right.y is hardcoded 0 (no roll), so neither has a field here.
 * cone_norm_factor = sqrt(k²·(1+k²)) is pre-computed client-side so HW
 * can recover the cone surface |normal| with one mul. Sun is derived from
 * (lx, ly, lz) - no separate sun fields. */
struct frame_inputs {
    int32_t lx, ly, lz;
    int32_t sx, sy, sz;
    uint32_t max_depth;
    uint32_t shift_0;
    uint32_t shift_1;
    int32_t cam_x, cam_z;
    int32_t right_x, right_z;
    int32_t up_x, up_y, up_z;
    int32_t fwd_x, fwd_y, fwd_z;
    int32_t r0_sq, r1_sq;
    int32_t sc0_x, sc0_y, sc0_z;
    int32_t pc1_r, pc1_g, pc1_b;
    int32_t pc2_r, pc2_g, pc2_b;
    int32_t scol0_r, scol0_g, scol0_b;
    int32_t scol1_r, scol1_g, scol1_b;
    uint32_t floor_mode;
    int32_t floor_lim_x, floor_lim_z;
    int32_t cone_x, cone_y, cone_z;
    int32_t cone_k_sq;
    int32_t cone_height;
    int32_t cone_col_r, cone_col_g, cone_col_b;
    uint32_t reflect_shift_cone;
    int32_t cone_norm_factor;
};

static inline void kick_frame(volatile struct rtb_csrs *csr,
                              const struct frame_inputs *in)
{
    csr->light_x        = (uint32_t)in->lx;
    csr->light_y        = (uint32_t)in->ly;
    csr->light_z        = (uint32_t)in->lz;
    csr->sc1_x          = (uint32_t)in->sx;
    csr->sc1_y          = (uint32_t)in->sy;
    csr->sc1_z          = (uint32_t)in->sz;
    csr->max_depth      = in->max_depth & 0x7u;
    csr->reflect_shift_0 = in->shift_0 & 0x1fu;
    csr->reflect_shift_1 = in->shift_1 & 0x1fu;
```

```
    csr->cam_x          = (uint32_t)in->cam_x;
    csr->cam_z          = (uint32_t)in->cam_z;
    csr->right_x        = (uint32_t)in->right_x;
    csr->right_z        = (uint32_t)in->right_z;
    csr->up_x           = (uint32_t)in->up_x;
    csr->up_y           = (uint32_t)in->up_y;
    csr->up_z           = (uint32_t)in->up_z;
    csr->fwd_x          = (uint32_t)in->fwd_x;
    csr->fwd_y          = (uint32_t)in->fwd_y;
    csr->fwd_z          = (uint32_t)in->fwd_z;
    csr->r0_sq          = (uint32_t)in->r0_sq;
    csr->r1_sq          = (uint32_t)in->r1_sq;
    /* Visual-upgrades CSRs -(#0#5). Same atomic commit as the rest. */
    csr->sc0_x          = (uint32_t)in->sc0_x;
    csr->sc0_y          = (uint32_t)in->sc0_y;
    csr->sc0_z          = (uint32_t)in->sc0_z;
    csr->pc1_r          = (uint32_t)in->pc1_r;
    csr->pc1_g          = (uint32_t)in->pc1_g;
    csr->pc1_b          = (uint32_t)in->pc1_b;
    csr->pc2_r          = (uint32_t)in->pc2_r;
    csr->pc2_g          = (uint32_t)in->pc2_g;
    csr->pc2_b          = (uint32_t)in->pc2_b;
    csr->scol0_r        = (uint32_t)in->scol0_r;
    csr->scol0_g        = (uint32_t)in->scol0_g;
    csr->scol0_b        = (uint32_t)in->scol0_b;
    csr->scol1_r        = (uint32_t)in->scol1_r;
    csr->scol1_g        = (uint32_t)in->scol1_g;
    csr->scol1_b        = (uint32_t)in->scol1_b;
    csr->floor_mode     = in->floor_mode & 0x3u;
    csr->floor_lim_x    = (uint32_t)in->floor_lim_x;
    csr->floor_lim_z    = (uint32_t)in->floor_lim_z;
    /* Cone primitive */
    csr->cone_x         = (uint32_t)in->cone_x;
    csr->cone_y         = (uint32_t)in->cone_y;
    csr->cone_z         = (uint32_t)in->cone_z;
    csr->cone_k_sq      = (uint32_t)in->cone_k_sq;
    csr->cone_height    = (uint32_t)in->cone_height;
    csr->cone_col_r     = (uint32_t)in->cone_col_r;
    csr->cone_col_g     = (uint32_t)in->cone_col_g;
    csr->cone_col_b     = (uint32_t)in->cone_col_b;
    csr->reflect_shift_cone = in->reflect_shift_cone & 0x1fu;
    csr->cone_norm_factor   = (uint32_t)in->cone_norm_factor;
    csr->control            = RTB_CONTROL_FRAME_START;
}

static int recv_frame_inputs(int fd, struct frame_inputs *in)
{
    uint32_t hdr[49];
    if (recv_exact(fd, hdr, sizeof(hdr)) < 0)
        return -1;
    in->lx        = (int32_t)ntohl(hdr[0]);
    in->ly        = (int32_t)ntohl(hdr[1]);
    in->lz        = (int32_t)ntohl(hdr[2]);
    in->sx        = (int32_t)ntohl(hdr[3]);
    in->sy        = (int32_t)ntohl(hdr[4]);
    in->sz        = (int32_t)ntohl(hdr[5]);
    in->max_depth = ntohl(hdr[6]);
    in->shift_0   = ntohl(hdr[7]);
    in->shift_1   = ntohl(hdr[8]);
    in->cam_x     = (int32_t)ntohl(hdr[9]);
    in->cam_z     = (int32_t)ntohl(hdr[10]);
    in->right_x   = (int32_t)ntohl(hdr[11]);
    in->right_z   = (int32_t)ntohl(hdr[12]);
    in->up_x      = (int32_t)ntohl(hdr[13]);
    in->up_y      = (int32_t)ntohl(hdr[14]);
    in->up_z      = (int32_t)ntohl(hdr[15]);
    in->fwd_x     = (int32_t)ntohl(hdr[16]);
    in->fwd_y     = (int32_t)ntohl(hdr[17]);
    in->fwd_z     = (int32_t)ntohl(hdr[18]);
    in->r0_sq     = (int32_t)ntohl(hdr[19]);
    in->r1_sq     = (int32_t)ntohl(hdr[20]);
    in->sc0_x     = (int32_t)ntohl(hdr[21]);
    in->sc0_y     = (int32_t)ntohl(hdr[22]);
    in->sc0_z     = (int32_t)ntohl(hdr[23]);
```

```
    in->pc1_r     = (int32_t)ntohl(hdr[24]);
    in->pc1_g     = (int32_t)ntohl(hdr[25]);
    in->pc1_b     = (int32_t)ntohl(hdr[26]);
    in->pc2_r     = (int32_t)ntohl(hdr[27]);
    in->pc2_g     = (int32_t)ntohl(hdr[28]);
    in->pc2_b     = (int32_t)ntohl(hdr[29]);
    in->scol0_r   = (int32_t)ntohl(hdr[30]);
    in->scol0_g   = (int32_t)ntohl(hdr[31]);
    in->scol0_b   = (int32_t)ntohl(hdr[32]);
    in->scol1_r   = (int32_t)ntohl(hdr[33]);
    in->scol1_g   = (int32_t)ntohl(hdr[34]);
    in->scol1_b   = (int32_t)ntohl(hdr[35]);
    in->floor_mode = ntohl(hdr[36]);
    in->floor_lim_x = (int32_t)ntohl(hdr[37]);
    in->floor_lim_z = (int32_t)ntohl(hdr[38]);
    in->cone_x      = (int32_t)ntohl(hdr[39]);
    in->cone_y      = (int32_t)ntohl(hdr[40]);
    in->cone_z      = (int32_t)ntohl(hdr[41]);
    in->cone_k_sq   = (int32_t)ntohl(hdr[42]);
    in->cone_height = (int32_t)ntohl(hdr[43]);
    in->cone_col_r  = (int32_t)ntohl(hdr[44]);
    in->cone_col_g  = (int32_t)ntohl(hdr[45]);
    in->cone_col_b  = (int32_t)ntohl(hdr[46]);
    in->reflect_shift_cone = ntohl(hdr[47]);
    in->cone_norm_factor = (int32_t)ntohl(hdr[48]);
    return 0;
}

/* Drain the in-flight HW frame so DMA doesn't write to the buffer after
 * the userspace mmap is torn down. Called on client disconnect. */
static void drain_frame(volatile struct rtb_csrs *csr)
{
    while (!(csr->status & RTB_STATUS_FRAME_DONE))
        ;
    csr->control = RTB_CONTROL_CLEAR_FRAME_DONE;
}

static void handle_client(int dev_fd,
                          volatile struct rtb_csrs *csr,
                          const uint32_t *frame,
                          int cli_fd,
                          struct sockaddr_in *addr)
{
    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &addr->sin_addr, ip, sizeof(ip));
    printf("client connected: %s:%d\n", ip, ntohs(addr->sin_port));

    struct frame_inputs in0;
    struct frame_inputs in_pending;

    if (recv_frame_inputs(cli_fd, &in0) < 0 ||
        recv_frame_inputs(cli_fd, &in_pending) < 0) {
        printf("client disconnected during prime\n");
        close(cli_fd);
        return;
    }

    kick_frame(csr, &in0);

    unsigned frames = 0;

    for (;;) {
        struct timespec t0, t_wait, t_kick, t_send;
        if (verbose) clock_gettime(CLOCK_MONOTONIC, &t0);

        uint32_t done_slot;
        if (ioctl(dev_fd, RTB_IOCTL_WAIT_FRAME, &done_slot) < 0) {
            perror("ioctl(RTB_IOCTL_WAIT_FRAME)");
            drain_frame(csr);
            break;
        }
        if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_wait);

        /* Kick N+1 before sending N so HW renders in parallel with TCP. */
```

```
        kick_frame(csr, &in_pending);
        if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_kick);

        int send_failed = 0;
        if (send_all(cli_fd, frame + done_slot * SLOT_WORDS,
                     SLOT_BYTES) < 0) {
            perror("send");
            send_failed = 1;
        }
        if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_send);

        if (send_failed) {
            drain_frame(csr);
            break;
        }

        if (verbose) {
            #define DELTA_MS(a, b) (((b).tv_sec - (a).tv_sec) * 1000.0 + \
                                    ((b).tv_nsec - (a).tv_nsec) / 1.0e6)
            double total_ms = DELTA_MS(t0, t_send);
            double wait_ms  = DELTA_MS(t0, t_wait);
            double kick_ms  = DELTA_MS(t_wait, t_kick);
            double send_ms  = DELTA_MS(t_kick, t_send);
            #undef DELTA_MS

            fprintf(stderr,
                    "frame %u: %.2f ms (%.1f fps) - wait=%.2f kick=%.3f send=%.2f\n",
                    frames, total_ms, 1000.0 / total_ms,
                    wait_ms, kick_ms, send_ms);
        }

        frames++;

        if (recv_frame_inputs(cli_fd, &in_pending) < 0) {
            drain_frame(csr);
            break;
        }
    }

    printf("client disconnected (%u frames served)\n", frames);
    close(cli_fd);
}

static void usage(const char *prog)
{
    fprintf(stderr, "usage: %s [-v] [port]\n  -v  per-frame timing log\n", prog);
}

int main(int argc, char *argv[])
{
    uint16_t port = DEFAULT_PORT;
    int opt_c;
    while ((opt_c = getopt(argc, argv, "vh")) != -1) {
        switch (opt_c) {
        case 'v': verbose = 1; break;
        case 'h':
        default:  usage(argv[0]); return opt_c == 'h' ? 0 : 1;
        }
    }
    if (optind < argc)
        port = (uint16_t)atoi(argv[optind]);

    signal(SIGPIPE, SIG_IGN);
    signal(SIGINT, sigint_handler);

    int dev_fd = open("/dev/raytracer", O_RDWR);
    if (dev_fd < 0) {
        perror("open /dev/raytracer");
        return 1;
    }

    struct rtb_layout layout;
    if (ioctl(dev_fd, RTB_IOCTL_GET_LAYOUT, &layout) < 0) {
        perror("ioctl(RTB_IOCTL_GET_LAYOUT)");
```

```
        close(dev_fd);
        return 1;
    }

    volatile struct rtb_csrs *csr = mmap(NULL, layout.csr_size,
                                         PROT_READ | PROT_WRITE,
                                         MAP_SHARED, dev_fd,
                                         layout.csr_offset);
    if (csr == MAP_FAILED) {
        perror("mmap CSR");
        close(dev_fd);
        return 1;
    }

    /* Covers both slots; slot N starts at frame[N * SLOT_WORDS]. */
    const uint32_t *frame = mmap(NULL, layout.frame_buf_bytes,
                                 PROT_READ, MAP_SHARED, dev_fd,
                                 layout.frame_buf_offset);
    if (frame == MAP_FAILED) {
        perror("mmap frame");
        close(dev_fd);
        return 1;
    }

    int listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd < 0) {
        perror("socket");
        close(dev_fd);
        return 1;
    }

    int opt = 1;
    setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    struct sockaddr_in saddr;
    memset(&saddr, 0, sizeof(saddr));
    saddr.sin_family      = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port        = htons(port);

    if (bind(listen_fd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
        perror("bind");
        close(listen_fd);
        close(dev_fd);
        return 1;
    }

    if (listen(listen_fd, 1) < 0) {
        perror("listen");
        close(listen_fd);
        close(dev_fd);
        return 1;
    }

    printf("listening on :%d  (%dx%d, %d bytes/frame, pipeline-depth-2)\n",
           port, WIDTH, HEIGHT, FRAME_BYTES);

    while (!stop) {
        struct sockaddr_in caddr;
        socklen_t clen = sizeof(caddr);
        int cli_fd = accept(listen_fd, (struct sockaddr *)&caddr, &clen);
        if (cli_fd < 0) {
            if (errno == EINTR)
                break;
            perror("accept");
            continue;
        }

        opt = 1;
        setsockopt(cli_fd, IPPROTO_TCP, TCP_NODELAY, &opt, sizeof(opt));

        /* SO_SNDBUF  frame so send() returns once the kernel has copied
         * from the cached mmap, allowing wire-out to overlap HW render. */
        int sndbuf = 1 << 20;
```

```
        setsockopt(cli_fd, SOL_SOCKET, SO_SNDBUF, &sndbuf, sizeof(sndbuf));

        handle_client(dev_fd, csr, frame, cli_fd, &caddr);
    }

    printf("\nshutting down\n");
    close(listen_fd);
    munmap((void *)frame, layout.frame_buf_bytes);
    munmap((void *)csr, layout.csr_size);
    close(dev_fd);
    return 0;
}
```

SW/fpga/render_frame.c

```
/*
 * render_frame.c - one-shot tool: render one full 480×360 frame via the
 * HW per-frame auto-kick (CONTROL[2] = FRAME_START), then write it as a
 * raw PPM image to stdout.
 *
 * Usage:
 *   ./render_frame LX LY LZ [MAX_DEPTH SHIFT0 SHIFT1] > frame.ppm
 *
 * Light position (3 floats, e.g. 0.0 4.0 4.0) is required. Reflection
 * params are optional; when omitted the HW boots into its default
 * scene (max_depth=3, shift0=2, shift1=1 → blue sphere reflects 25%,
 * red sphere 50%, 3 bounces deep). Pass `MAX_DEPTH=0` to disable
 * reflection entirely.
 *
 * Output is a P6 PPM file (viewable with feh, eog, GIMP, etc.).
 *
 * The frame buffer is a 2-slot ring (1.4 MB total). RTB_IOCTL_WAIT_FRAME
 * blocks until FRAME_DONE, invalidates the just-finished slot's caches,
 * and returns its index - we read directly from the cached mmap. No
 * pipelining (single frame, doesn't pay).
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/mman.h>

#include "../raytracer_batch.h"

#define WIDTH    480
#define HEIGHT   360
#define FP_SCALE 8192   /* 1 << FRAC_BITS at Q14.13 (must match raytracer.sv FRAC_BITS) */

/* HW reset defaults (must match raytracer.sv DEFAULT_MAX_DEPTH /
 * DEFAULT_REFLECT_SHIFT_*). */
#define DEFAULT_MAX_DEPTH        3u
#define DEFAULT_REFLECT_SHIFT_0  2u
#define DEFAULT_REFLECT_SHIFT_1  1u

int main(int argc, char *argv[])
{
    if (argc != 4 && argc != 7) {
        fprintf(stderr,
            "usage: %s LX LY LZ [MAX_DEPTH SHIFT0 SHIFT1]\n"
            "       light position required (3 floats);\n"
            "       reflection params optional, defaulting to\n"
            "       depth=%u shift0=%u shift1=%u (HW reset values).\n",
            argv[0], DEFAULT_MAX_DEPTH,
            DEFAULT_REFLECT_SHIFT_0, DEFAULT_REFLECT_SHIFT_1);
        return 1;
    }

    int32_t lx = (int32_t)(atof(argv[1]) * FP_SCALE);
```

```
    int32_t ly = (int32_t)(atof(argv[2]) * FP_SCALE);
    int32_t lz = (int32_t)(atof(argv[3]) * FP_SCALE);

    uint32_t max_depth = DEFAULT_MAX_DEPTH;
    uint32_t shift_0   = DEFAULT_REFLECT_SHIFT_0;
    uint32_t shift_1   = DEFAULT_REFLECT_SHIFT_1;
    if (argc == 7) {
        max_depth = (uint32_t)strtoul(argv[4], NULL, 0);
        shift_0   = (uint32_t)strtoul(argv[5], NULL, 0);
        shift_1   = (uint32_t)strtoul(argv[6], NULL, 0);
    }

    int fd = open("/dev/raytracer", O_RDWR);
    if (fd < 0) {
        perror("open /dev/raytracer");
        return 1;
    }

    struct rtb_layout layout;
    if (ioctl(fd, RTB_IOCTL_GET_LAYOUT, &layout) < 0) {
        perror("ioctl(RTB_IOCTL_GET_LAYOUT)");
        close(fd);
        return 1;
    }

    volatile struct rtb_csrs *csr = mmap(NULL, layout.csr_size,
                                         PROT_READ | PROT_WRITE,
                                         MAP_SHARED, fd,
                                         layout.csr_offset);
    if (csr == MAP_FAILED) {
        perror("mmap CSR");
        close(fd);
        return 1;
    }

    const uint32_t *frame = mmap(NULL, layout.frame_buf_bytes,
                                 PROT_READ, MAP_SHARED, fd,
                                 layout.frame_buf_offset);
    if (frame == MAP_FAILED) {
        perror("mmap frame");
        close(fd);
        return 1;
    }

    /* Stage light + reflection params + kick. sc1_* is left untouched
     * here (keeps its current value, which is the HW default at first
     * kick). HW picks the write_slot internally; we get the just-finished
     * slot back from RTB_IOCTL_WAIT_FRAME. */
    csr->light_x         = (uint32_t)lx;
    csr->light_y         = (uint32_t)ly;
    csr->light_z         = (uint32_t)lz;
    csr->max_depth       = max_depth & 0x7u;
    csr->reflect_shift_0 = shift_0 & 0x1fu;
    csr->reflect_shift_1 = shift_1 & 0x1fu;
    csr->control         = RTB_CONTROL_FRAME_START;

    uint32_t slot;
    if (ioctl(fd, RTB_IOCTL_WAIT_FRAME, &slot) < 0) {
        perror("ioctl(RTB_IOCTL_WAIT_FRAME)");
        close(fd);
        return 1;
    }

    /* Unpack →BGRXRGB directly from the cached mmap. The WAIT_FRAME
     * ioctl already invalidated this slot's cache lines, so the loop
     * fills L1/L2 from DDR in one streaming pass. */
    const uint32_t *slot_src = frame + slot * (WIDTH * HEIGHT);

    /* PPM header */
    printf("P6\n%d %d\n255\n", WIDTH, HEIGHT);

    uint8_t row[WIDTH * 3];
    for (unsigned y = 0; y < HEIGHT; y++) {
        const uint32_t *row_src = slot_src + y * WIDTH;
```

```
        uint8_t *p = row;
        for (unsigned x = 0; x < WIDTH; x++) {
            uint32_t pk = row_src[x];
            *p++ = (uint8_t)((pk >> 16) & 0xff);
            *p++ = (uint8_t)((pk >>  8) & 0xff);
            *p++ = (uint8_t)( pk        & 0xff);
        }
        fwrite(row, 1, sizeof(row), stdout);
    }

    munmap((void *)frame, layout.frame_buf_bytes);
    munmap((void *)csr, layout.csr_size);
    close(fd);

    fprintf(stderr,
        "rendered %dx%d frame (light %.1f, %.1f, %.1f, "
        "max_depth=%u shift0=%u shift1=%u)\n",
        WIDTH, HEIGHT,
        lx / (double)FP_SCALE,
        ly / (double)FP_SCALE,
        lz / (double)FP_SCALE,
        max_depth, shift_0, shift_1);
    return 0;
}
```

SW/fpga/Makefile

```
CROSS_COMPILE ?=
CC      := $(CROSS_COMPILE)gcc
CFLAGS  := -O2 -Wall -Wextra -I..
LDFLAGS :=

all: server render_frame

server: server.c ../raytracer_batch.h
    $(CC) $(CFLAGS) -o $@ server.c $(LDFLAGS)

render_frame: render_frame.c ../raytracer_batch.h
    $(CC) $(CFLAGS) -o $@ render_frame.c $(LDFLAGS)

clean:
    rm -f server render_frame

.PHONY: clean
```

3 Desktop Client + Demos

SW/desktop/client.py

```
#!/usr/bin/env python3
"""Desktop client for the FPGA raytracer.

Single frame:
    python client.py HOST PORT --light LX LY LZ -o out.png

Animation (orbiting light, saved to GIF):
    python client.py HOST PORT --animate N -o out.gif [--radius 4.0] [--height 4.0] [--duration 80]

Live stream (orbiting light, displayed in a window):
    python client.py HOST PORT --stream [--radius 4.0] [--height 4.0] [--period 3.0]
"""

import argparse
import math
import socket
import struct
import time

from PIL import Image
```

```

# Frame geometry - must match the FPGA hardware / driver constants.
WIDTH = 480
HEIGHT = 360
FP_SHIFT = 13
FP_SCALE = 1 << FP_SHIFT
# Wire format is the raw HW DDR layout: 4 bytes per pixel as BGRX.
# FPGA writes {00,R,G,B} per word; little-endian ARM HPS stores it as
# B,G,R,00 in memory and ships those bytes straight to the desktop,
# which decodes →BGRXRGB with PIL (cheaper than unpacking on the
# Cortex-A9).
FRAME_BYTES = WIDTH * HEIGHT * 4  # 691,200

# HW defaults - must mirror raytracer.sv's DEFAULT_* localparams so a
# client that doesn't override anything sends the byte-equivalent scene.
SPHERE0_CENTER = (0.0, 1.0, 0.0)
SPHERE1_DEFAULT_CENTER = (2.5, 1.0, 0.0)

DEFAULT_MAX_DEPTH       = 3   # 0..7; 0 disables reflection
DEFAULT_REFLECT_SHIFT_0 = 2   # blue sphere reflection scale = >> 2 (25%)
DEFAULT_REFLECT_SHIFT_1 = 1   # red sphere                   = >> 1 (50%)

# SW pre-squares radii before shipping; the CSR holds r² directly so HW
# avoids a multiply per pixel.
DEFAULT_RADIUS_0 = 1.0
DEFAULT_RADIUS_1 = 1.0

# Camera defaults. cam_y is locked at 1.5 in HW (no Y-translation), so
# only X / Z are CSR-driven. yaw=0, pitch=0 produces the identity basis
# (right=x̂, up=ŷ, fwd=ẑ) - the legacy hardcoded scene byte-equal.
DEFAULT_CAM_X     = 0.0
DEFAULT_CAM_Z     = -8.0
DEFAULT_YAW       = 0.0
DEFAULT_PITCH     = 0.0

DEFAULT_SC0        = (0.0, 1.0, 0.0)
DEFAULT_PC1_RGB    = (0.9, 0.9, 0.9)
DEFAULT_PC2_RGB    = (0.4, 0.4, 0.4)
DEFAULT_SCOL0_RGB  = (0.2, 0.3, 0.8)
DEFAULT_SCOL1_RGB  = (0.8, 0.3, 0.2)
DEFAULT_FLOOR_MODE = 0
DEFAULT_FLOOR_LIM  = 1000.0       # ~infinite half-extent

# Y-axis cone, apex on top, opens downward over y   [apex.y - height,
# apex.y]. height=0 disables the cone - matches the HW reset state.
DEFAULT_CONE_APEX     = (0.0, 2.5, 0.0)
DEFAULT_CONE_K_SQ     = 1.0       # tan²(half-angle); 1.0 = 45°
DEFAULT_CONE_HEIGHT   = 0.0
DEFAULT_CONE_COL_RGB  = (0.5, 0.5, 0.5)
DEFAULT_REFLECT_SHIFT_CONE = 0


def to_fp(f: float) -> int:
    """Convert a float to Q14.13 signed fixed-point (int32)."""
    return int(round(f * FP_SCALE))


def camera_basis_from_euler(yaw: float, pitch: float
                            ) -> tuple[tuple[float, float],
                                       tuple[float, float, float],
                                       tuple[float, float, float]]:
    """Compute the right / up / fwd basis vectors from yaw + pitch
    (radians). right.y is always 0 (we don't support roll), so right
    is returned as a 2-tuple (x, z); up and fwd are full 3-tuples.

    Convention (matches raytracer_fp.py:camera_basis):
      yaw=0, pitch=0   →  right=(-1, 0), up=(0, 1, 0), fwd=(0, 0, 1)
      yaw>0            →  camera turns right (forward picks up +x)
      pitch>0          →  camera tilts up    (forward picks up +y)
    """
    cy, sy = math.cos(yaw),   math.sin(yaw)
    cp, sp = math.cos(pitch), math.sin(pitch)
    right   = (-cy,                sy)
    up      = (-sy * sp,  cp,     -cy * sp)
```

```
    forward = ( sy * cp,  sp,      cy * cp)
    return right, up, forward


def pack_inputs(lx: float, ly: float, lz: float,
                sx: float, sy: float, sz: float,
                max_depth: int = DEFAULT_MAX_DEPTH,
                shift_0: int = DEFAULT_REFLECT_SHIFT_0,
                shift_1: int = DEFAULT_REFLECT_SHIFT_1,
                cam_x: float = DEFAULT_CAM_X,
                cam_z: float = DEFAULT_CAM_Z,
                yaw: float = DEFAULT_YAW,
                pitch: float = DEFAULT_PITCH,
                r0: float = DEFAULT_RADIUS_0,
                r1: float = DEFAULT_RADIUS_1,
                sc0: tuple[float, float, float] = DEFAULT_SC0,
                pc1_rgb: tuple[float, float, float] = DEFAULT_PC1_RGB,
                pc2_rgb: tuple[float, float, float] = DEFAULT_PC2_RGB,
                scol0_rgb: tuple[float, float, float] = DEFAULT_SCOL0_RGB,
                scol1_rgb: tuple[float, float, float] = DEFAULT_SCOL1_RGB,
                floor_mode: int = DEFAULT_FLOOR_MODE,
                floor_lim_x: float = DEFAULT_FLOOR_LIM,
                floor_lim_z: float = DEFAULT_FLOOR_LIM,
                cone_apex: tuple[float, float, float] = DEFAULT_CONE_APEX,
                cone_k_sq: float = DEFAULT_CONE_K_SQ,
                cone_height: float = DEFAULT_CONE_HEIGHT,
                cone_col_rgb: tuple[float, float, float] = DEFAULT_CONE_COL_RGB,
                reflect_shift_cone: int = DEFAULT_REFLECT_SHIFT_CONE,
                ) -> bytes:
    """Pack a per-frame inputs blob: 49 × int32 (network-order, 196 bytes).
    Layout matches server.c's recv_frame_inputs():
      [0..2]   light XYZ                (Q14.13 signed)
      [3..5]   sphere-1 center XYZ
      [6]      max_depth                (0..7; 0 disables reflection)
      [7]      reflect_shift_0          (0..31)
      [8]      reflect_shift_1          (0..31)
      [9..10]  camera origin X / Z      (cam_y locked in HW)
      [11..12] basis right.x / right.z  (right.y = 0)
      [13..15] basis up.x / .y / .z
      [16..18] basis fwd.x / .y / .z
      [19..20] sphere radii² (r0², r1²) - SW pre-squares r
      [21..23] sphere-0 center XYZ
      [24..26] floor color #1 RGB
      [27..29] floor color #2 RGB
      [30..32] sphere-0 RGB
      [33..35] sphere-1 RGB
      [36]     floor_mode               (0..3)
      [37..38] floor_lim X / Z
      [39..41] cone apex XYZ
      [42]     cone_k_sq                (tan²(half-angle))
      [43]     cone_height              (0 disables)
      [44..46] cone color RGB
      [47]     reflect_shift_cone       (0..31)
      [48]     cone_norm_factor         (SW-computed √(k²·(1+k²)) so HW
                                         recovers |normal| with one mul)
    HW masks integer fields to their register widths on commit, so
    out-of-range values are silently clamped."""
    right, up, forward = camera_basis_from_euler(yaw, pitch)
    cone_norm_factor = math.sqrt(cone_k_sq * (1.0 + cone_k_sq))
    return struct.pack(
        "!iiiiiiIIIiiiiiiiiiii"     # legacy 21
        "iiiiiiiiiiiiiiIii"         # extension 18 (1 unsigned: floor_mode)
        "iiiiiiiIii",               # cone     10 (1 unsigned: reflect_shift_cone)
        to_fp(lx), to_fp(ly), to_fp(lz),
        to_fp(sx), to_fp(sy), to_fp(sz),
        max_depth & 0x7,
        shift_0 & 0x1f,
        shift_1 & 0x1f,
        to_fp(cam_x),     to_fp(cam_z),
        to_fp(right[0]),  to_fp(right[1]),
        to_fp(up[0]),     to_fp(up[1]),    to_fp(up[2]),
        to_fp(forward[0]), to_fp(forward[1]), to_fp(forward[2]),
        to_fp(r0 * r0),   to_fp(r1 * r1),
        to_fp(sc0[0]), to_fp(sc0[1]), to_fp(sc0[2]),
```

```
        to_fp(pc1_rgb[0]), to_fp(pc1_rgb[1]), to_fp(pc1_rgb[2]),
        to_fp(pc2_rgb[0]), to_fp(pc2_rgb[1]), to_fp(pc2_rgb[2]),
        to_fp(scol0_rgb[0]), to_fp(scol0_rgb[1]), to_fp(scol0_rgb[2]),
        to_fp(scol1_rgb[0]), to_fp(scol1_rgb[1]), to_fp(scol1_rgb[2]),
        floor_mode & 0x3,
        to_fp(floor_lim_x), to_fp(floor_lim_z),
        to_fp(cone_apex[0]), to_fp(cone_apex[1]), to_fp(cone_apex[2]),
        to_fp(cone_k_sq), to_fp(cone_height),
        to_fp(cone_col_rgb[0]), to_fp(cone_col_rgb[1]), to_fp(cone_col_rgb[2]),
        reflect_shift_cone & 0x1f,
        to_fp(cone_norm_factor),
    )


def recv_frame(sock: socket.socket) -> bytes:
    """Read a full frame of RGB888 data off the socket."""
    buf = bytearray(FRAME_BYTES)
    view = memoryview(buf)
    got = 0
    while got < FRAME_BYTES:
        n = sock.recv_into(view[got:])
        if n == 0:
            raise ConnectionError(
                f"server closed after {got}/{FRAME_BYTES} bytes"
            )
        got += n
    return bytes(buf)


# Wire protocol - pipeline-depth-2.
# The server kicks frame N+1 to HW before reading frame N's slot, so HW
# renders in parallel with SW reading + sending. The client must therefore
# send 2 inputs blobs up front ("prime") and then 1 blob per received
# frame. For one-shot single-frame rendering: prime with 2 copies of the
# same blob, receive 1 frame, and close - the server drains the extra
# in-flight frame on disconnect.


def request_frame(sock: socket.socket,
                  lx: float, ly: float, lz: float,
                  sphere: tuple[float, float, float] = SPHERE1_DEFAULT_CENTER,
                  max_depth: int = DEFAULT_MAX_DEPTH,
                  shift_0: int = DEFAULT_REFLECT_SHIFT_0,
                  shift_1: int = DEFAULT_REFLECT_SHIFT_1) -> bytes:
    """One-shot: prime with 2 identical inputs blobs, recv 1 frame. The
    extra in-flight HW frame is drained on close. For continuous use see
    render_animation / render_stream, which keep the pipe full."""
    payload = pack_inputs(lx, ly, lz, *sphere,
                          max_depth=max_depth,
                          shift_0=shift_0, shift_1=shift_1)
    sock.sendall(payload)
    sock.sendall(payload)
    return recv_frame(sock)


def decode(data: bytes) -> Image.Image:
    """Decode raw BGRX (4 B/pixel) bytes into an RGB PIL Image - PIL's
    raw decoder does the →BGRXRGB swap in one C-level pass."""
    return Image.frombuffer("RGB", (WIDTH, HEIGHT), data, "raw", "BGRX", 0, 1)


def render_single(sock, lx, ly, lz, output,
                  max_depth=DEFAULT_MAX_DEPTH,
                  shift_0=DEFAULT_REFLECT_SHIFT_0,
                  shift_1=DEFAULT_REFLECT_SHIFT_1):
    """Render one frame and save it."""
    data = request_frame(sock, lx, ly, lz,
                         max_depth=max_depth,
                         shift_0=shift_0, shift_1=shift_1)
    img = decode(data)
    img.save(output)
    print(f"saved {output} ({WIDTH}x{HEIGHT}, "
          f"max_depth={max_depth} shifts=({shift_0},{shift_1}))")

```

```
243
244 class LightController:
245     """Interface for driving the light position over time."""
246
247     def light_at(self, t: float) -> tuple[float, float, float]:
248         raise NotImplementedError
249
250
251 class AutoOrbitController(LightController):
252     """Orbits the light in a circle at constant height."""
253
254     def __init__(self, radius: float = 4.0, height: float = 4.0, period: float = 3.0):
255         self.radius = radius
256         self.height = height
257         self.period = period
258
259     def light_at(self, t: float) -> tuple[float, float, float]:
260         angle = 2.0 * math.pi * (t / self.period)
261         return (
262             self.radius * math.cos(angle),
263             self.height,
264             self.radius * math.sin(angle),
265         )
266
267
268 class SphereController:
269     """Interface for driving sphere 1 (red) center position over time."""
270
271     def sphere_at(self, t: float) -> tuple[float, float, float]:
272         raise NotImplementedError
273
274
275 class StaticSphereController(SphereController):
276     """Holds sphere 1 at a fixed position (matches the HW DEFAULT_SC1_*)."""
277
278     def __init__(self, position: tuple[float, float, float] = SPHERE1_DEFAULT_CENTER):
279         self.position = position
280
281     def sphere_at(self, t: float) -> tuple[float, float, float]:
282         return self.position
283
284
285 class OrbitSphereController(SphereController):
286     """Orbits sphere 1 (red) horizontally around a fixed center (default:
287     sphere 0's position) at constant height."""
288
289     def __init__(self,
290                  center: tuple[float, float, float] = SPHERE0_CENTER,
291                  radius: float = 2.5,
292                  height: float = 1.0,
293                  period: float = 40.0):
294         self.center = center
295         self.radius = radius
296         self.height = height
297         self.period = period
298
299     def sphere_at(self, t: float) -> tuple[float, float, float]:
300         angle = 2.0 * math.pi * (t / self.period)
301         cx, _, cz = self.center
302         return (
303             cx + self.radius * math.cos(angle),
304             self.height,
305             cz + self.radius * math.sin(angle),
306         )
307
308
309 def render_stream(sock: socket.socket,
310                   light_controller: LightController,
311                   sphere_controller: SphereController,
312                   max_depth: int = DEFAULT_MAX_DEPTH,
313                   shift_0: int = DEFAULT_REFLECT_SHIFT_0,
314                   shift_1: int = DEFAULT_REFLECT_SHIFT_1) -> None:
315     """Continuously request frames and blit them to a pygame window.
316
```

```
    Uses the Stage 4 pipeline-depth-2 protocol: prime with inputs 0 and 1,
    then send inputs N+2 after receiving frame N. Light + sphere positions
    are sampled at fixed wall-clock intervals so both orbits advance
    smoothly even when the pipeline is N=2 frames ahead of the displayed
    frame. Reflection params are constant across the stream."""
    try:
        import pygame
    except ImportError:
        raise SystemExit(
            "pygame is required for --stream mode. Install it with: pip install pygame"
        )

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.FULLSCREEN | pygame.SCALED)
    pygame.display.set_caption("FPGA Raytracer (live)")
    clock = pygame.time.Clock()

    pygame.joystick.init()
    if pygame.joystick.get_count() > 0:
        joy = pygame.joystick.Joystick(0)
        joy.init()
        print(f"Connected: {joy.get_name()}")
    else:
        print("ERROR: No joysticks found!")
        return

    t0 = time.monotonic()

    # Light position is driven by joystick axes 0/1 (X/Z). The
    # light_controller arg is kept in the signature for API compatibility
    # with the main()→render_stream path but the joystick takes priority.
    light_pos = [0.0, 4.0, 0.0]
    LIGHT_MOVE_SCALE = 5.0

    def user_inputs() -> bytes:
        t = time.monotonic() - t0
        return pack_inputs(
            *light_pos,
            *sphere_controller.sphere_at(t),
            max_depth=max_depth,
            shift_0=shift_0, shift_1=shift_1)

    def update_light_pos():
        light_pos[0] = joy.get_axis(0) * LIGHT_MOVE_SCALE
        light_pos[2] = -joy.get_axis(1) * LIGHT_MOVE_SCALE

    sock.sendall(user_inputs())
    sock.sendall(user_inputs())

    frames = 0
    running = True
    try:
        while running:
            for ev in pygame.event.get():
                if ev.type == pygame.QUIT:
                    running = False
                    break
                if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
                    running = False
                    break
            if not running:
                break

            data = recv_frame(sock)
            update_light_pos()
            # Pipeline next inputs immediately so they land on the wire
            # while the server is still sending the rest of this frame.
            sock.sendall(user_inputs())

            img = decode(data)
            surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
            screen.blit(surf, (0, 0))
            pygame.display.flip()
```

```
            frames += 1
            clock.tick()
            if frames % 30 == 0:
                print(f"{frames} frames  ({clock.get_fps():.1f} fps)")
    finally:
        pygame.quit()
        print(f"streamed {frames} frames")


def render_animation(sock, num_frames, output, radius, height, duration,
                     sphere_controller: SphereController | None = None,
                     max_depth: int = DEFAULT_MAX_DEPTH,
                     shift_0: int = DEFAULT_REFLECT_SHIFT_0,
                     shift_1: int = DEFAULT_REFLECT_SHIFT_1):
    """Render an orbiting-light animation and save as GIF. Optional
    sphere_controller orbits sphere 1; defaults to a static position.
    Same pipeline-depth-2 wire protocol as render_stream."""
    if sphere_controller is None:
        sphere_controller = StaticSphereController()

    def light_for(i):
        angle = 2.0 * math.pi * i / num_frames
        return (radius * math.cos(angle), height, radius * math.sin(angle))

    # Sample sphere at the same per-frame phase as light. Picking the
    # period from duration*num_frames makes a full sphere orbit span
    # the whole GIF.
    def inputs_for(i):
        return pack_inputs(
            *light_for(i),
            *sphere_controller.sphere_at(i * duration / 1000.0),
            max_depth=max_depth,
            shift_0=shift_0, shift_1=shift_1,
        )

    sock.sendall(inputs_for(0))
    if num_frames >= 2:
        sock.sendall(inputs_for(1))
    else:
        sock.sendall(inputs_for(0))   # 1-frame anim: pad the prime

    # Server's loop ships frame K then blocks on inputs K+2 before iter
    # K+1. To ship the final real frame we send N+1 blobs total: the 2
    # primes, N-2 in-loop sends, and one phantom tail (duplicate of
    # inputs N-1) so iter N-2's tail recv unblocks. The server then
    # kicks a phantom frame N which is drained on close.
    frames = []
    for i in range(num_frames):
        data = recv_frame(sock)
        next_idx = i + 2
        if next_idx < num_frames:
            sock.sendall(inputs_for(next_idx))
        elif next_idx == num_frames:
            sock.sendall(inputs_for(num_frames - 1))   # phantom tail
        frames.append(decode(data))
        print(f"frame {i + 1}/{num_frames}")

    frames[0].save(
        output,
        save_all=True,
        append_images=frames[1:],
        duration=duration,
        loop=0,
    )
    print(f"saved {output} ({num_frames} frames)")


def main():
    parser = argparse.ArgumentParser(description="FPGA raytracer client")
    parser.add_argument("host", help="FPGA server IP or hostname")
    parser.add_argument("port", type=int, help="FPGA server port")
    parser.add_argument("-o", "--output", default="out.png", help="output file")

    group = parser.add_mutually_exclusive_group(required=True)
```

```
    group.add_argument(
        "--light",
        nargs=3,
        type=float,
        metavar=("LX", "LY", "LZ"),
        help="light position (floats); saves a single PNG",
    )
    group.add_argument(
        "--animate",
        type=int,
        metavar="N",
        help="render N frames with orbiting light; saves a GIF",
    )
    group.add_argument(
        "--stream",
        action="store_true",
        help="continuously display orbiting-light frames in a pygame window",
    )

    parser.add_argument("--radius", type=float, default=4.0, help="orbit radius")
    parser.add_argument("--height", type=float, default=4.0, help="light height")
    parser.add_argument(
        "--duration", type=int, default=80, help="GIF frame duration (ms)"
    )
    parser.add_argument(
        "--period", type=float, default=3.0,
        help="stream mode: seconds per full light orbit",
    )

    parser.add_argument(
        "--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
        help=f"reflection recursion cap (0..7; default {DEFAULT_MAX_DEPTH}; 0 disables)",
    )
    parser.add_argument(
        "--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0,
        help=f"sphere 0 reflectivity = >> shift (0..31; default {DEFAULT_REFLECT_SHIFT_0})",
    )
    parser.add_argument(
        "--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1,
        help=f"sphere 1 reflectivity = >> shift (0..31; default {DEFAULT_REFLECT_SHIFT_1})",
    )

    args = parser.parse_args()

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((args.host, args.port))

    try:
        if args.light is not None:
            render_single(sock, *args.light, args.output,
                          max_depth=args.max_depth,
                          shift_0=args.shift0, shift_1=args.shift1)
        elif args.stream:
            light_controller = AutoOrbitController(
                radius=args.radius, height=args.height, period=args.period,
            )
            sphere_controller = OrbitSphereController()
            render_stream(sock, light_controller, sphere_controller,
                          max_depth=args.max_depth,
                          shift_0=args.shift0, shift_1=args.shift1)
        else:
            render_animation(
                sock, args.animate, args.output,
                args.radius, args.height, args.duration,
                sphere_controller=OrbitSphereController(),
                max_depth=args.max_depth,
                shift_0=args.shift0, shift_1=args.shift1,
            )
    finally:
        sock.close()


if __name__ == "__main__":
    main()
```

SW/desktop/camera.py

```
#!/usr/bin/env python3
"""Desktop client for the FPGA raytracer with joystick CAMERA control.

Like client.py's --stream mode but the joystick drives the camera
instead of the light:
    Axis 0  → strafe (camera-relative ±right)
    Axis 1  → walk   (camera-relative ±forward)
    Axis 2  → yaw    (clockwise from above)        - if 4-axis device
    Axis 3  → pitch  (clamped ±80° to dodge basis singularity)

Axes 2/3 use a wide cliff (default 30%% of full range) anchored at the
absolute axis zero, NOT a runtime snapshot. On a flight stick (where
axis 3 is a throttle slider that doesn't self-center) this means: park
the slider near its mechanical center to disable pitch; leaving it
parked off-center will integrate pitch continuously. Move the throttle
slider into the cliff to stop pitching.

The camera moves in its own XZ frame so "forward" always means whichever
direction the camera is looking at, not world +Z. cam_y is locked at 1.5
in HW (no Y-translation), matching the camera plan.

Light stays at a static position (--light, default (0, 4, 0)) so camera
motion is the only thing changing in the scene - easier to tell whether
the basis math is firing correctly.

Usage:
    python camera.py HOST PORT
    python camera.py HOST PORT --start-x 2.0 --start-yaw 30
    python camera.py HOST PORT --light 4 4 4 --max-depth 0  # no reflection
"""

import argparse
import math
import socket
import struct
import time

from PIL import Image

import client

# Frame geometry - must match the FPGA hardware / driver constants.
WIDTH = 480
HEIGHT = 360
FP_SHIFT = 13
FP_SCALE = 1 << FP_SHIFT
FRAME_BYTES = WIDTH * HEIGHT * 4

SPHERE0_CENTER = (0.0, 1.0, 0.0)
SPHERE1_DEFAULT_CENTER = (2.5, 1.0, 0.0)

DEFAULT_MAX_DEPTH       = 3
DEFAULT_REFLECT_SHIFT_0 = 2
DEFAULT_REFLECT_SHIFT_1 = 1

DEFAULT_CAM_X     = 0.0
DEFAULT_CAM_Z     = -8.0
DEFAULT_YAW       = 0.0
DEFAULT_PITCH     = 0.0

# Pitch limit: at  ±/2 the basis collapses (fwd → ŷ±, up degenerates).
# 80° is well clear of the singularity and still gives near-vertical look.
PITCH_LIMIT = math.radians(80.0)


def camera_basis_from_euler(yaw: float, pitch: float):
    """Same convention as client.py / raytracer_fp.camera_basis. yaw=0,
    pitch=0 yields right=(-1,0), up=(0,1,0), fwd=(0,0,1). Kept local for
    CameraController's strafing physics; the wire-format basis goes
    through client.pack_inputs."""
    cy, sy = math.cos(yaw),   math.sin(yaw)
    cp, sp = math.cos(pitch), math.sin(pitch)
```

```
    right   = (-cy,                sy)
    up      = (-sy * sp,  cp,     -cy * sp)
    forward = ( sy * cp,  sp,      cy * cp)
    return right, up, forward


def pack_inputs(lx: float, ly: float, lz: float,
                sx: float, sy: float, sz: float,
                max_depth: int, shift_0: int, shift_1: int,
                cam_x: float, cam_z: float,
                yaw: float, pitch: float,
                r0: float = 1.0, r1: float = 1.0,
                ) -> bytes:
    return client.pack_inputs(
        lx, ly, lz, sx, sy, sz,
        max_depth=max_depth, shift_0=shift_0, shift_1=shift_1,
        cam_x=cam_x, cam_z=cam_z, yaw=yaw, pitch=pitch,
        r0=r0, r1=r1,
    )


def recv_frame(sock: socket.socket) -> bytes:
    buf = bytearray(FRAME_BYTES)
    view = memoryview(buf)
    got = 0
    while got < FRAME_BYTES:
        n = sock.recv_into(view[got:])
        if n == 0:
            raise ConnectionError(f"server closed after {got}/{FRAME_BYTES} bytes")
        got += n
    return bytes(buf)


def decode(data: bytes) -> Image.Image:
    return Image.frombuffer("RGB", (WIDTH, HEIGHT), data, "raw", "BGRX", 0, 1)


class CameraController:
    """Joystick-driven camera state, integrated each frame against dt.

    Translation is camera-relative on the XZ ground plane (pitch is not
    applied to walk - the camera doesn't fly). Strafe direction is the
    player's right when looking +fwd, derived from yaw alone - this is
    the player-strafe basis, NOT the ray-direction basis from
    camera_basis_from_euler (those differ by a world-x flip from the
     screenworld mapping).
    """

    def __init__(self, joy,
                 x: float = DEFAULT_CAM_X, z: float = DEFAULT_CAM_Z,
                 yaw: float = DEFAULT_YAW, pitch: float = DEFAULT_PITCH,
                 move_scale: float = 3.0,
                 rot_scale_deg: float = 90.0,
                 deadzone: float = 0.15,
                 rot_deadzone: float = 0.30,
                 invert_yaw: bool = True,
                 invert_pitch: bool = False,
                 invert_strafe: bool = True):
        self.joy = joy
        self.x = x
        self.z = z
        self.yaw = yaw
        self.pitch = pitch
        self.move_scale = move_scale
        self.rot_scale = math.radians(rot_scale_deg)
        # Wider rotation cliff because flight-stick twist/throttle don't
        # self-center the way the spring-loaded translation axes do.
        self.deadzone     = deadzone
        self.rot_deadzone = rot_deadzone
        # Default sign flips picked to match Logitech Extreme 3D Pro feel:
        # twist CW → camera right, stick right → walk toward screen-right.
        self.yaw_sign    = -1.0 if invert_yaw    else 1.0
        self.pitch_sign  = -1.0 if invert_pitch  else 1.0
        self.strafe_sign = -1.0 if invert_strafe else 1.0
```

```
        self.has_right_stick = joy.get_numaxes() >= 4

    def _dz(self, v: float, dz: float = None) -> float:
        """Deadzone with linear remap so partial deflection is still useful."""
        if dz is None:
            dz = self.deadzone
        if abs(v) < dz:
            return 0.0
        sign = 1.0 if v > 0.0 else -1.0
        return sign * (abs(v) - dz) / (1.0 - dz)

    def update(self, dt: float) -> None:
        lx = self._dz(self.joy.get_axis(0))
        ly = self._dz(self.joy.get_axis(1))
        if self.has_right_stick:
            rx = self._dz(self.joy.get_axis(2), self.rot_deadzone)
            ry = self._dz(self.joy.get_axis(3), self.rot_deadzone)
        else:
            rx = ry = 0.0

        self.yaw   += rx * self.rot_scale * dt * self.yaw_sign
        self.pitch += -ry * self.rot_scale * dt * self.pitch_sign
        if self.pitch >  PITCH_LIMIT: self.pitch =  PITCH_LIMIT
        if self.pitch < -PITCH_LIMIT: self.pitch = -PITCH_LIMIT

        sy, cy = math.sin(self.yaw), math.cos(self.yaw)
        walk   = -ly
        strafe =  lx * self.strafe_sign

        self.x += (sy * walk + cy * strafe) * self.move_scale * dt
        self.z += (cy * walk - sy * strafe) * self.move_scale * dt


def render_stream(sock: socket.socket,
                  light: tuple[float, float, float],
                  sphere: tuple[float, float, float],
                  start: tuple[float, float, float, float],
                  move_scale: float,
                  rot_scale_deg: float,
                  rot_deadzone: float,
                  invert_yaw: bool,
                  invert_pitch: bool,
                  invert_strafe: bool,
                  max_depth: int,
                  shift_0: int,
                  shift_1: int,
                  r0_start: float,
                  r1_start: float,
                  radius_rate: float,
                  radius_min: float,
                  radius_max: float,
                  btn_r0_grow: int,
                  btn_r0_shrink: int,
                  btn_r1_grow: int,
                  btn_r1_shrink: int) -> None:
    """Pipeline-depth-2 stream loop with joystick camera. Same prime +
    steady-state recv/send pattern as client.render_stream."""
    try:
        import pygame
    except ImportError:
        raise SystemExit(
            "pygame is required for camera.py. Install it with: pip install pygame"
        )

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.FULLSCREEN | pygame.SCALED)
    pygame.display.set_caption("FPGA Raytracer (camera)")
    clock = pygame.time.Clock()

    pygame.joystick.init()
    if pygame.joystick.get_count() == 0:
        print("ERROR: No joysticks found!")
        return
    joy = pygame.joystick.Joystick(0)
```

```
    joy.init()
    print(f"Connected: {joy.get_name()}  ({joy.get_numaxes()} axes)")

    sx, sz, syaw, spitch = start
    camera = CameraController(joy, x=sx, z=sz, yaw=syaw, pitch=spitch,
                              move_scale=move_scale,
                              rot_scale_deg=rot_scale_deg,
                              rot_deadzone=rot_deadzone,
                              invert_yaw=invert_yaw,
                              invert_pitch=invert_pitch,
                              invert_strafe=invert_strafe)
    if camera.has_right_stick:
        print(f"axis2/3 cliff: ±{camera.rot_deadzone:.2f} around 0  "
              f"(park throttle/twist at center for no rotation)")

    nbuttons = joy.get_numbuttons()
    radius_buttons_ok = (
        max(btn_r0_grow, btn_r0_shrink, btn_r1_grow, btn_r1_shrink) < nbuttons
    )
    if not radius_buttons_ok:
        print(f"WARN: joystick has {nbuttons} buttons; radius buttons "
              f"({btn_r0_grow}/{btn_r0_shrink}/{btn_r1_grow}/{btn_r1_shrink}) "
              "out of range - radii will stay at start values.")
    else:
        print(f"radii: hold btn{btn_r0_grow}/{btn_r0_shrink} for r0 grow/shrink, "
              f"btn{btn_r1_grow}/{btn_r1_shrink} for r1 ({radius_rate:.2f} u/s, "
              f"clamped [{radius_min:.2f}, {radius_max:.2f}])")

    r0 = max(radius_min, min(radius_max, r0_start))
    r1 = max(radius_min, min(radius_max, r1_start))

    def make_inputs() -> bytes:
        return pack_inputs(*light, *sphere,
                           max_depth, shift_0, shift_1,
                           camera.x, camera.z, camera.yaw, camera.pitch,
                           r0, r1)

    sock.sendall(make_inputs())
    sock.sendall(make_inputs())

    frames = 0
    last_t = time.monotonic()
    running = True
    try:
        while running:
            for ev in pygame.event.get():
                if ev.type == pygame.QUIT:
                    running = False
                    break
                if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
                    running = False
                    break
            if not running:
                break

            data = recv_frame(sock)

            now = time.monotonic()
            dt = now - last_t
            last_t = now
            camera.update(dt)

            # Hold-to-grow/shrink radii at radius_rate while a button is held.
            if radius_buttons_ok:
                if joy.get_button(btn_r0_grow):
                    r0 = min(radius_max, r0 + radius_rate * dt)
                if joy.get_button(btn_r0_shrink):
                    r0 = max(radius_min, r0 - radius_rate * dt)
                if joy.get_button(btn_r1_grow):
                    r1 = min(radius_max, r1 + radius_rate * dt)
                if joy.get_button(btn_r1_shrink):
                    r1 = max(radius_min, r1 - radius_rate * dt)

            sock.sendall(make_inputs())
```

```
            img = decode(data)
            surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
            screen.blit(surf, (0, 0))
            pygame.display.flip()

            frames += 1
            clock.tick()
            if frames % 30 == 0:
                print(f"{frames} frames  ({clock.get_fps():.1f} fps)  "
                      f"cam=({camera.x:+.2f}, 1.50, {camera.z:+.2f})  "
                      f"yaw={math.degrees(camera.yaw):+6.1f}°  "
                      f"pitch={math.degrees(camera.pitch):+5.1f}°  "
                      f"r0={r0:.2f} r1={r1:.2f}")
    finally:
        pygame.quit()
        print(f"streamed {frames} frames")


def main():
    parser = argparse.ArgumentParser(
        description="FPGA raytracer client - joystick drives camera"
    )
    parser.add_argument("host", help="FPGA server IP or hostname")
    parser.add_argument("port", type=int, help="FPGA server port")

    parser.add_argument("--light", nargs=3, type=float,
                        default=[0.0, 4.0, 0.0],
                        metavar=("LX", "LY", "LZ"),
                        help="static light position (default: 0 4 0)")
    parser.add_argument("--sphere1", nargs=3, type=float,
                        default=list(SPHERE1_DEFAULT_CENTER),
                        metavar=("SX", "SY", "SZ"),
                        help=f"static sphere-1 center (default: {SPHERE1_DEFAULT_CENTER})")

    parser.add_argument("--start-x",     type=float, default=DEFAULT_CAM_X)
    parser.add_argument("--start-z",     type=float, default=DEFAULT_CAM_Z)
    parser.add_argument("--start-yaw",   type=float, default=0.0,
                        help="starting yaw in degrees (default 0)")
    parser.add_argument("--start-pitch", type=float, default=0.0,
                        help="starting pitch in degrees (default 0)")

    parser.add_argument("--move-scale", type=float, default=3.0,
                        help="translation speed at full deflection (units/sec)")
    parser.add_argument("--rot-scale",  type=float, default=90.0,
                        help="rotation speed at full deflection (deg/sec)")
    parser.add_argument("--rot-deadzone", type=float, default=0.30,
                        help="cliff/deadzone for axes 2/3 (yaw/pitch). Larger "
                             "= more dead area around the launch zero before "
                             "rotation starts integrating. Default 0.30 = 30%% "
                             "of full range either side.")
    parser.add_argument("--no-invert-yaw", action="store_true",
                        help="don't flip yaw sign. Default IS to invert "
                             "(matches Logitech Extreme 3D Pro twist convention).")
    parser.add_argument("--invert-pitch", action="store_true",
                        help="flip pitch sign (default: not inverted).")
    parser.add_argument("--no-invert-strafe", action="store_true",
                        help="don't flip strafe sign. Default IS to invert "
                             "(HW basis right=(-cos(yaw),sin(yaw)) mirrors "
                             "world-x to screen-x; flipping makes stick-right "
                             "move the camera toward screen-right).")

    parser.add_argument("--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
                        help=f"reflection recursion cap (0..7; default {DEFAULT_MAX_DEPTH})")
    parser.add_argument("--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0)
    parser.add_argument("--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1)

    parser.add_argument("--r0", type=float, default=1.0,
                        help="starting radius for sphere 0 (blue). Default 1.0.")
    parser.add_argument("--r1", type=float, default=1.0,
                        help="starting radius for sphere 1 (red). Default 1.0.")
    parser.add_argument("--radius-rate", type=float, default=1.0,
                        help="grow/shrink rate while a radius button is held "
                             "(units/sec). Default 1.0.")
```

```
    parser.add_argument("--radius-min", type=float, default=0.2,
                        help="lower clamp on radii (default 0.2).")
    parser.add_argument("--radius-max", type=float, default=4.0,
                        help="upper clamp on radii (default 4.0).")
    parser.add_argument("--btn-r0-grow",   type=int, default=5,
                        help="joystick button to grow sphere-0 radius (default 0)")
    parser.add_argument("--btn-r0-shrink", type=int, default=3,
                        help="joystick button to shrink sphere-0 radius (default 1)")
    parser.add_argument("--btn-r1-grow",   type=int, default=4,
                        help="joystick button to grow sphere-1 radius (default 2)")
    parser.add_argument("--btn-r1-shrink", type=int, default=2,
                        help="joystick button to shrink sphere-1 radius (default 3)")

    args = parser.parse_args()

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((args.host, args.port))
    try:
        render_stream(
            sock,
            light=tuple(args.light),
            sphere=tuple(args.sphere1),
            start=(args.start_x, args.start_z,
                   math.radians(args.start_yaw),
                   math.radians(args.start_pitch)),
            move_scale=args.move_scale,
            rot_scale_deg=args.rot_scale,
            rot_deadzone=args.rot_deadzone,
            invert_yaw=not args.no_invert_yaw,
            invert_pitch=args.invert_pitch,
            invert_strafe=not args.no_invert_strafe,
            max_depth=args.max_depth,
            shift_0=args.shift0,
            shift_1=args.shift1,
            r0_start=args.r0,
            r1_start=args.r1,
            radius_rate=args.radius_rate,
            radius_min=args.radius_min,
            radius_max=args.radius_max,
            btn_r0_grow=args.btn_r0_grow,
            btn_r0_shrink=args.btn_r0_shrink,
            btn_r1_grow=args.btn_r1_grow,
            btn_r1_shrink=args.btn_r1_shrink,
        )
    finally:
        sock.close()


if __name__ == "__main__":
    main()
```

SW/desktop/pong.py

```
#!/usr/bin/env python3
"""Pong, rendered by the FPGA raytracer.

Two spheres are paddles and the cone is the ball, played in the XY
plane in front of the default camera:

    Sphere 0 (blue)  -> LEFT  paddle (player)
    Sphere 1 (red)   -> RIGHT paddle (AI)
    Cone     (white) -> ball

Controls (joystick, like camera_cone.py):
    Axis 1     -> player paddle up/down (stick up = paddle up)
    Button 0   -> serve / next round / restart after game over
    ESC        -> quit

If no joystick is connected the demo falls back to AI-vs-AI: the
"player" paddle also tracks the ball, and serves / restarts trigger
automatically after a short pause so the screen never sits idle.

```

```
The wire / prime-the-pipeline pattern matches camera_cone.py: send 2
input blobs up front, then 1 per received frame. Game state is
integrated against wall-clock dt between recv_frame calls.

Usage:
    python pong.py HOST PORT
"""

import argparse
import math
import random
import socket
import time

import client
import camera


# Court geometry (world units). Paddles sit at x = ±PADDLE_X; ball is
# the cone primitive so BALL_R is the collision radius, separate from
# the rendered cone height.
PADDLE_X     = 3.0
PADDLE_R     = 0.6
BALL_R       = 0.35
COURT_Y_MIN  = 0.5
COURT_Y_MAX  = 4.0
COURT_X_OUT  = 4.5

BALL_CONE_HEIGHT = 0.7
BALL_CONE_K_SQ   = 1.0           # 45° half-angle → base radius = height
BALL_COLOR       = (0.95, 0.95, 0.95)

PLAYER_SPEED    = 4.5
AI_SPEED        = 3.0            # < PLAYER_SPEED so the AI is beatable
BALL_SPEED_INIT = 3.5
BALL_SPEED_MAX  = 7.5

WIN_SCORE = 5

LIGHT_POS = (1.5, 4.5, -2.0)
CAM_X     = 0.0
CAM_Z     = -8.0
CAM_YAW   = 0.0
CAM_PITCH = math.radians(8.0)    # tilt up so ceiling isn't off-screen

STATE_SERVING  = "serving"
STATE_PLAYING  = "playing"
STATE_GAMEOVER = "gameover"

PADDLE_Y_MIN = COURT_Y_MIN + PADDLE_R
PADDLE_Y_MAX = COURT_Y_MAX - PADDLE_R
BALL_Y_MIN   = COURT_Y_MIN + BALL_R
BALL_Y_MAX   = COURT_Y_MAX - BALL_R


class PongGame:
    def __init__(self):
        mid = 0.5 * (COURT_Y_MIN + COURT_Y_MAX)
        self.player_y = mid
        self.ai_y     = mid
        self.ball_x   = 0.0
        self.ball_y   = mid
        self.vx       = 0.0
        self.vy       = 0.0
        self.score_p  = 0
        self.score_a  = 0
        self.state    = STATE_SERVING
        self._set_serve_velocity(direction=-1)

    def _set_serve_velocity(self, direction: int):
        angle = random.uniform(-0.30, 0.30) * math.pi
        self.vx = direction * BALL_SPEED_INIT * math.cos(angle)
        self.vy =             BALL_SPEED_INIT * math.sin(angle)

```

```
    def _reset_ball(self, direction: int):
        self.ball_x = 0.0
        self.ball_y = 0.5 * (COURT_Y_MIN + COURT_Y_MAX)
        self._set_serve_velocity(direction)

    def serve(self):
        if self.state == STATE_SERVING:
            self.state = STATE_PLAYING
        elif self.state == STATE_GAMEOVER:
            self.score_p = 0
            self.score_a = 0
            self._reset_ball(direction=-1)
            self.state = STATE_PLAYING

    def update_player(self, axis: float, dt: float):
        # joystick up reports negative; negate so stick-up moves the paddle up.
        self.player_y -= axis * PLAYER_SPEED * dt
        if self.player_y < PADDLE_Y_MIN: self.player_y = PADDLE_Y_MIN
        if self.player_y > PADDLE_Y_MAX: self.player_y = PADDLE_Y_MAX

    def _track_ball(self, paddle_y: float, dt: float) -> float:
        delta = self.ball_y - paddle_y
        step  = AI_SPEED * dt
        if abs(delta) <= step:
            paddle_y = self.ball_y
        else:
            paddle_y += step if delta > 0 else -step
        if paddle_y < PADDLE_Y_MIN: paddle_y = PADDLE_Y_MIN
        if paddle_y > PADDLE_Y_MAX: paddle_y = PADDLE_Y_MAX
        return paddle_y

    def _update_ai(self, dt: float):
        self.ai_y = self._track_ball(self.ai_y, dt)

    def update_player_ai(self, dt: float):
        self.player_y = self._track_ball(self.player_y, dt)

    def _paddle_hit(self, paddle_y: float, dir_x: int):
        # Reflect x, add english from where the ball hit on the paddle,
        # bump speed slightly each rally (capped at BALL_SPEED_MAX).
        rel   = (self.ball_y - paddle_y) / PADDLE_R
        rel   = max(-1.0, min(1.0, rel))
        speed = math.hypot(self.vx, self.vy) * 1.05
        if speed > BALL_SPEED_MAX:
            speed = BALL_SPEED_MAX
        angle    = rel * math.radians(55.0)
        self.vx  = dir_x * speed * math.cos(angle)
        self.vy  =         speed * math.sin(angle)

    def step(self, dt: float):
        if self.state != STATE_PLAYING:
            return

        self._update_ai(dt)

        self.ball_x += self.vx * dt
        self.ball_y += self.vy * dt

        if self.ball_y < BALL_Y_MIN:
            self.ball_y = BALL_Y_MIN
            self.vy = abs(self.vy)
        elif self.ball_y > BALL_Y_MAX:
            self.ball_y = BALL_Y_MAX
            self.vy = -abs(self.vy)

        # Gate on vx sign so a ball that already passed the paddle plane
        # can't re-collide on the way out.
        rsum2 = (PADDLE_R + BALL_R) ** 2
        if self.vx < 0:
            dx = self.ball_x - (-PADDLE_X)
            dy = self.ball_y - self.player_y
            if dx > 0 and dx * dx + dy * dy < rsum2:
                self._paddle_hit(self.player_y, +1)
        elif self.vx > 0:
```

```
            dx = self.ball_x - PADDLE_X
            dy = self.ball_y - self.ai_y
            if dx < 0 and dx * dx + dy * dy < rsum2:
                self._paddle_hit(self.ai_y, -1)

        if self.ball_x < -COURT_X_OUT:
            self.score_a += 1
            self._after_score(direction=+1)
        elif self.ball_x > COURT_X_OUT:
            self.score_p += 1
            self._after_score(direction=-1)

    def _after_score(self, direction: int):
        if self.score_p >= WIN_SCORE or self.score_a >= WIN_SCORE:
            self.state = STATE_GAMEOVER
        else:
            self.state = STATE_SERVING
        self._reset_ball(direction)


def run(sock):
    try:
        import pygame
    except ImportError:
        raise SystemExit("pygame required: pip install pygame")

    pygame.init()
    screen = pygame.display.set_mode((camera.WIDTH, camera.HEIGHT), pygame.SCALED)
    pygame.display.set_caption("FPGA Raytracer Pong")
    clock = pygame.time.Clock()
    pygame.font.init()
    big_font = pygame.font.SysFont("monospace", 36, bold=True)
    sm_font  = pygame.font.SysFont("monospace", 14, bold=True)

    pygame.joystick.init()
    joy = None
    if pygame.joystick.get_count() > 0:
        joy = pygame.joystick.Joystick(0)
        joy.init()
        print(f"Connected: {joy.get_name()}  ({joy.get_numaxes()} axes)")
    else:
        print("No joystick - running AI-vs-AI demo mode")

    # No-joystick mode auto-serves on these delays so the screen never
    # idles on the "press button" prompt.
    AUTO_SERVE_DELAY    = 1.0
    AUTO_RESTART_DELAY  = 3.0
    auto_serve_at = None

    game = PongGame()

    def make_inputs() -> bytes:
        sc0 = (-PADDLE_X, game.player_y, 0.0)
        sc1 = ( PADDLE_X, game.ai_y,     0.0)
        # Apex sits half a cone-height above the logical ball position so
        # the rendered cone body straddles the collider symmetrically.
        cone_apex = (game.ball_x, game.ball_y + 0.5 * BALL_CONE_HEIGHT, 0.0)
        return client.pack_inputs(
            *LIGHT_POS, *sc1,
            max_depth=2, shift_0=2, shift_1=2,
            cam_x=CAM_X, cam_z=CAM_Z,
            yaw=CAM_YAW, pitch=CAM_PITCH,
            r0=PADDLE_R, r1=PADDLE_R,
            sc0=sc0,
            cone_apex=cone_apex,
            cone_k_sq=BALL_CONE_K_SQ,
            cone_height=BALL_CONE_HEIGHT,
            cone_col_rgb=BALL_COLOR,
            reflect_shift_cone=0,
        )

    sock.sendall(make_inputs())
    sock.sendall(make_inputs())

```

```
    last_t = time.monotonic()
    running = True
    try:
        while running:
            for ev in pygame.event.get():
                if ev.type == pygame.QUIT:
                    running = False
                    break
                if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
                    running = False
                    break
                if ev.type == pygame.JOYBUTTONDOWN and ev.button == 0:
                    game.serve()
                if ev.type == pygame.KEYDOWN and ev.key == pygame.K_SPACE:
                    game.serve()
            if not running:
                break

            data = camera.recv_frame(sock)

            now = time.monotonic()
            dt  = now - last_t
            if dt > 0.05:           # clamp so a network stall doesn't teleport the ball
                dt = 0.05
            last_t = now

            if joy is not None:
                axis = joy.get_axis(1)
                if abs(axis) < 0.15:
                    axis = 0.0
                game.update_player(axis, dt)
            else:
                game.update_player_ai(dt)
                if game.state == STATE_PLAYING:
                    auto_serve_at = None
                else:
                    delay = AUTO_RESTART_DELAY if game.state == STATE_GAMEOVER \
                                               else AUTO_SERVE_DELAY
                    if auto_serve_at is None:
                        auto_serve_at = now + delay
                    elif now >= auto_serve_at:
                        game.serve()
                        auto_serve_at = None
            game.step(dt)

            sock.sendall(make_inputs())

            img  = camera.decode(data)
            surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
            screen.blit(surf, (0, 0))

            score_txt = f"{game.score_p}   {game.score_a}"
            score_surf   = big_font.render(score_txt, True, (255, 255, 255))
            score_shadow = big_font.render(score_txt, True, (0, 0, 0))
            sx = camera.WIDTH // 2 - score_surf.get_width() // 2
            screen.blit(score_shadow, (sx + 2, 10))
            screen.blit(score_surf,   (sx,     8))

            if game.state == STATE_SERVING:
                msg = "serving..." if joy is None else "press button 0 to serve"
            elif game.state == STATE_GAMEOVER:
                winner = "PLAYER" if game.score_p > game.score_a else "AI"
                tail   = "restarting..." if joy is None else "press button 0"
                msg = f"{winner} WINS  -  {tail}"
            else:
                msg = None
            if msg is not None:
                m_surf   = sm_font.render(msg, True, (255, 255, 255))
                m_shadow = sm_font.render(msg, True, (0, 0, 0))
                mx = camera.WIDTH // 2 - m_surf.get_width() // 2
                my = camera.HEIGHT - 28
                screen.blit(m_shadow, (mx + 1, my + 1))
                screen.blit(m_surf,   (mx,     my))

```

```
            fps_txt = f"{clock.get_fps():5.1f} fps"
            f_surf   = sm_font.render(fps_txt, True, (255, 255, 255))
            f_shadow = sm_font.render(fps_txt, True, (0, 0, 0))
            screen.blit(f_shadow, (7, 7))
            screen.blit(f_surf,   (6, 6))

            pygame.display.flip()
            clock.tick()
    finally:
        pygame.quit()


def main():
    ap = argparse.ArgumentParser(
        description="Pong, rendered by the FPGA raytracer"
    )
    ap.add_argument("host")
    ap.add_argument("port", type=int)
    args = ap.parse_args()

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((args.host, args.port))
    try:
        run(sock)
    finally:
        sock.close()


if __name__ == "__main__":
    main()
```

SW/desktop/sandbox.py

```
#!/usr/bin/env python3
"""Sandbox - two-flightstick puppet show that exposes every CSR knob.

Per-stick controls (stick 0 drives ball 0, stick 1 drives ball 1):

    Main stick (axes 0/1)
        default                 ball X/Z velocity (this stick's ball)
        +trigger (btn 0) held   camera X/Z velocity (twist = yaw)
        +side    (btn 10) held  camera pitch (stick fwd/back = look up/down)
        +thumb   (btn 1) held   stick 0: cone k^2 (X) + height (Y)
                                stick 1: cone apex X / apex Z
    Throttle (axis 3)           stick 0: cone reflectivity (up=bright, down=matte)
                                stick 1: ball radius velocity (additive w/ btns)
    POV hat                     stick 0: light X / Z
                                stick 1: light Y
    Top cluster
        btn 5 / 3               ball radius grow / shrink
        btn 4 / 2               ball Y up / down
    Base
        stick 0:
            btn 6               cycle floor mode (0..3)
            btn 7               cycle ball 0 color
            btn 8               toggle cone on/off
            btn 9               cycle max reflection depth (0..4)
        stick 1:
            btn 6               cycle ball 1 color
            btn 7               cycle cone color
            btn 8               cycle cone shape preset (k² + height)

Trigger and thumb are per-stick modifiers. Holding either stick's
trigger puts that stick into camera mode; the OTHER stick keeps doing
its default thing. The cone apex's y coordinate tracks the cone height
so the cone base stays on the floor.

Usage:
    python sandbox.py HOST PORT
"""

import argparse
```

```
import math
import socket
import time

import client
import camera


WIDTH  = camera.WIDTH
HEIGHT = camera.HEIGHT
CAM_Y  = 1.5

# Rates / scales - world units (or radians) per second.
BALL_MOVE_SCALE  = 3.0
BALL_Y_RATE      = 1.5
RADIUS_RATE      = 1.0
CAM_MOVE_SCALE   = 4.0
YAW_RATE         = math.radians(90.0)
PITCH_RATE       = math.radians(70.0)
PITCH_LIMIT      = math.radians(80.0)
LIGHT_MOVE_SCALE = 3.0
LIGHT_Y_RATE     = 1.5
CONE_HEIGHT_RATE = 1.5
CONE_K_RATE      = 0.6
CONE_APEX_RATE   = 2.0

BALL_Y_MIN, BALL_Y_MAX     = 0.3, 5.0
LIGHT_Y_MIN, LIGHT_Y_MAX   = 0.5, 8.0
R_MIN, R_MAX               = 0.2, 3.0
CONE_H_MIN, CONE_H_MAX     = 0.3, 5.0
CONE_K_MIN, CONE_K_MAX     = 0.05, 2.0
CAM_X_LIM, CAM_Z_LIM       = 15.0, 15.0

DZ_STICK    = 0.15
DZ_TWIST    = 0.30
DZ_THROTTLE = 0.15

BALL_COLORS = [
    (0.20, 0.30, 0.80),  (0.80, 0.30, 0.20),
    (0.20, 0.80, 0.30),  (0.95, 0.85, 0.20),
    (0.70, 0.30, 0.90),  (0.95, 0.55, 0.20),
    (0.20, 0.80, 0.80),  (0.95, 0.95, 0.95),
]
CONE_COLORS = [
    (0.95, 0.55, 0.20),
    (0.50, 0.50, 0.50),
    (0.20, 0.70, 0.30),
    (0.80, 0.20, 0.80),
    (0.30, 0.30, 0.90),
]

# (k_sq, height) presets. The pack step pins apex.y = height so the
# base always sits on the floor.
CONE_SHAPES = [
    (0.25, 3.0),   # narrow, tall
    (0.10, 4.0),   # spire
    (0.50, 2.5),   # standard
    (1.00, 1.5),   # 45° wide & short
    (2.00, 1.2),   # very wide cap
]

# reflect_shift_cone is a >> shift on the reflection contribution:
# 0 = brightest mirror;  8 = HW won't spawn a reflection ray (matte).
REFLECT_SHIFT_MIN = 0
REFLECT_SHIFT_MAX = 8

MAX_DEPTH_LEVELS = [0, 1, 2, 3, 4]


def deadzone(v: float, dz: float = DZ_STICK) -> float:
    if abs(v) < dz:
        return 0.0
    s = 1.0 if v > 0.0 else -1.0
    return s * (abs(v) - dz) / (1.0 - dz)
```

```


def clamp(x, lo, hi):
    return lo if x < lo else (hi if x > hi else x)


class State:
    def __init__(self):
        self.b0       = [-1.5, 1.0, 0.0]
        self.r0       = 1.0
        self.color0   = 0

        self.b1       = [ 1.5, 1.0, 0.0]
        self.r1       = 1.0
        self.color1   = 1

        self.light    = [3.0, 4.0, 2.0]

        self.cam_x    = 0.0
        self.cam_z    = -8.0
        self.yaw      = 0.0
        self.pitch    = math.radians(-3.0)

        self.cone_on            = False
        self.cone_apex_xz       = [0.0, 1.0]
        self.cone_height        = 3.0
        self.cone_k_sq          = 0.25
        self.cone_color         = 0
        self.cone_shape         = 0
        self.reflect_shift_cone = 2
        self.max_depth_idx      = 3

        self.floor_mode = 0

    def pack(self) -> bytes:
        # apex.y = height keeps the cone base on y = 0 (the floor plane).
        h = self.cone_height if self.cone_on else 0.0
        apex = (self.cone_apex_xz[0], h, self.cone_apex_xz[1])
        return client.pack_inputs(
            *self.light, *self.b1,
            max_depth=MAX_DEPTH_LEVELS[self.max_depth_idx],
            shift_0=1, shift_1=1,
            cam_x=self.cam_x, cam_z=self.cam_z,
            yaw=self.yaw, pitch=self.pitch,
            r0=self.r0, r1=self.r1,
            sc0=tuple(self.b0),
            scol0_rgb=BALL_COLORS[self.color0],
            scol1_rgb=BALL_COLORS[self.color1],
            floor_mode=self.floor_mode,
            cone_apex=apex,
            cone_k_sq=self.cone_k_sq,
            cone_height=h,
            cone_col_rgb=CONE_COLORS[self.cone_color],
            reflect_shift_cone=self.reflect_shift_cone,
        )


def _btn(joy, idx) -> bool:
    return idx < joy.get_numbuttons() and joy.get_button(idx) != 0


def _axis(joy, idx, dz=DZ_STICK) -> float:
    if idx >= joy.get_numaxes():
        return 0.0
    return deadzone(joy.get_axis(idx), dz)


def _hat(joy):
    if joy.get_numhats() == 0:
        return (0, 0)
    return joy.get_hat(0)


def update_stick(state: State, joy, joy_idx: int, dt: float):
```

```
    """Read one joystick's inputs and mutate state accordingly."""
    if joy is None:
        return

    mx = _axis(joy, 0)
    my = _axis(joy, 1)
    trigger = _btn(joy, 0)
    thumb   = _btn(joy, 1)
    side    = _btn(joy, 10)

    # Main stick: per-stick mode-switch on trigger / side / thumb.
    if trigger:
        sy, cy = math.sin(state.yaw), math.cos(state.yaw)
        walk   = -my
        strafe = -mx
        state.cam_x += (sy * walk + cy * strafe) * CAM_MOVE_SCALE * dt
        state.cam_z += (cy * walk - sy * strafe) * CAM_MOVE_SCALE * dt
        state.cam_x = clamp(state.cam_x, -CAM_X_LIM, CAM_X_LIM)
        state.cam_z = clamp(state.cam_z, -CAM_Z_LIM, CAM_Z_LIM)
        twist = _axis(joy, 2, DZ_TWIST)
        state.yaw += -twist * YAW_RATE * dt
    elif side:
        state.pitch += -my * PITCH_RATE * dt
        state.pitch = clamp(state.pitch, -PITCH_LIMIT, PITCH_LIMIT)
    elif thumb:
        if joy_idx == 0:
            state.cone_k_sq   += mx * CONE_K_RATE      * dt
            state.cone_height += -my * CONE_HEIGHT_RATE * dt
            state.cone_k_sq   = clamp(state.cone_k_sq,   CONE_K_MIN, CONE_K_MAX)
            state.cone_height = clamp(state.cone_height, CONE_H_MIN, CONE_H_MAX)
        else:
            state.cone_apex_xz[0] +=  mx * CONE_APEX_RATE * dt
            state.cone_apex_xz[1] += -my * CONE_APEX_RATE * dt
    else:
        ball = state.b0 if joy_idx == 0 else state.b1
        ball[0] += -mx * BALL_MOVE_SCALE * dt
        ball[2] += -my * BALL_MOVE_SCALE * dt

    hx, hy = _hat(joy)
    if joy_idx == 0:
        state.light[0] += hx * LIGHT_MOVE_SCALE * dt
        state.light[2] += hy * LIGHT_MOVE_SCALE * dt
    else:
        state.light[1] += hy * LIGHT_Y_RATE * dt
        state.light[1] = clamp(state.light[1], LIGHT_Y_MIN, LIGHT_Y_MAX)

    # Logitech flightstick throttle: axis 3 = -1 fully up, +1 fully down.
    throttle = _axis(joy, 3, DZ_THROTTLE)
    if joy_idx == 0:
        # Stick 0 throttle → cone reflect_shift (up = brightest mirror).
        norm = (throttle + 1.0) * 0.5
        shift = round(norm * (REFLECT_SHIFT_MAX - REFLECT_SHIFT_MIN)) + REFLECT_SHIFT_MIN
        if shift < REFLECT_SHIFT_MIN: shift = REFLECT_SHIFT_MIN
        if shift > REFLECT_SHIFT_MAX: shift = REFLECT_SHIFT_MAX
        state.reflect_shift_cone = shift
        dr_throttle = 0.0
    else:
        dr_throttle = -throttle * RADIUS_RATE * dt

    dr_btn = (1.0 if _btn(joy, 5) else 0.0) - (1.0 if _btn(joy, 3) else 0.0)
    dy_btn = (1.0 if _btn(joy, 4) else 0.0) - (1.0 if _btn(joy, 2) else 0.0)

    r_attr  = "r0" if joy_idx == 0 else "r1"
    new_r   = getattr(state, r_attr) + dr_throttle + dr_btn * RADIUS_RATE * dt
    setattr(state, r_attr, clamp(new_r, R_MIN, R_MAX))

    ball = state.b0 if joy_idx == 0 else state.b1
    ball[1] += dy_btn * BALL_Y_RATE * dt
    ball[1] = clamp(ball[1], BALL_Y_MIN, BALL_Y_MAX)


def handle_button_edge(ev, state: State):
    """Edge-triggered toggles / cycles on JOYBUTTONDOWN."""
    if ev.joy == 0:
```

```
        if   ev.button == 6: state.floor_mode = (state.floor_mode + 1) % 4
        elif ev.button == 7: state.color0     = (state.color0 + 1) % len(BALL_COLORS)
        elif ev.button == 8: state.cone_on    = not state.cone_on
        elif ev.button == 9:
            state.max_depth_idx = (state.max_depth_idx + 1) % len(MAX_DEPTH_LEVELS)
    elif ev.joy == 1:
        if   ev.button == 6: state.color1     = (state.color1 + 1) % len(BALL_COLORS)
        elif ev.button == 7: state.cone_color = (state.cone_color + 1) % len(CONE_COLORS)
        elif ev.button == 8:
            state.cone_shape = (state.cone_shape + 1) % len(CONE_SHAPES)
            state.cone_k_sq, state.cone_height = CONE_SHAPES[state.cone_shape]


def run(sock):
    try:
        import pygame
    except ImportError:
        raise SystemExit("pygame required: pip install pygame")

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.SCALED)
    pygame.display.set_caption("FPGA Raytracer - Sandbox")
    clock = pygame.time.Clock()
    pygame.font.init()
    sm = pygame.font.SysFont("monospace", 12, bold=True)

    pygame.joystick.init()
    joys = []
    for i in range(pygame.joystick.get_count()):
        j = pygame.joystick.Joystick(i)
        j.init()
        joys.append(j)
        print(f"Joystick {i}: {j.get_name()} ({j.get_numaxes()} axes, "
              f"{j.get_numbuttons()} buttons, {j.get_numhats()} hats)")
    if not joys:
        print("ERROR: no joysticks found - this demo requires at least one.")
        return

    state = State()
    sock.sendall(state.pack())
    sock.sendall(state.pack())

    last_t  = time.monotonic()
    running = True
    try:
        while running:
            for ev in pygame.event.get():
                if ev.type == pygame.QUIT:
                    running = False
                    break
                if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
                    running = False
                    break
                if ev.type == pygame.JOYBUTTONDOWN:
                    print(f"  joy{ev.joy} btn{ev.button}")
                    handle_button_edge(ev, state)
            if not running:
                break

            data = camera.recv_frame(sock)

            now = time.monotonic()
            dt  = now - last_t
            if dt > 0.05:
                dt = 0.05
            last_t = now

            for i, j in enumerate(joys):
                update_stick(state, j, i, dt)

            sock.sendall(state.pack())

            img  = camera.decode(data)
            surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
```

```
            screen.blit(surf, (0, 0))

            for i, j in enumerate(joys):
                if   _btn(j, 0):  mode = "CAM"
                elif _btn(j, 10): mode = "TILT"
                elif _btn(j, 1):  mode = "CONE"
                else:             mode = f"BALL{i}"
                held = [str(b) for b in range(j.get_numbuttons()) if j.get_button(b)]
                held_str = ",".join(held) if held else "-"
                msg = f"S{i}:{mode} held:{held_str}"
                surf_m  = sm.render(msg, True, (255, 255, 255))
                shadow  = sm.render(msg, True, (0, 0, 0))
                x = 6 + i * 180
                screen.blit(shadow,  (x + 1, 7))
                screen.blit(surf_m,  (x,     6))

            lines = [
                f"b0 ({state.b0[0]:+.1f},{state.b0[1]:.1f},{state.b0[2]:+.1f}) r={state.r0:.2f}",
                f"b1 ({state.b1[0]:+.1f},{state.b1[1]:.1f},{state.b1[2]:+.1f}) r={state.r1:.2f}",
                f"lt ({state.light[0]:+.1f},{state.light[1]:.1f},{state.light[2]:+.1f})",
                f"cm ({state.cam_x:+.1f},1.5,{state.cam_z:+.1f}) yaw={math.degrees(state.yaw):+5.0f}",
                f"cone {'ON' if state.cone_on else 'off'} h={state.cone_height:.1f} k2={state.cone_k_sq:.2f}",
                f"refl shift={state.reflect_shift_cone}  depth={MAX_DEPTH_LEVELS[state.max_depth_idx]}",
                f"floor {state.floor_mode}",
                f"{clock.get_fps():5.1f} fps",
            ]
            for k, line in enumerate(lines):
                s_surf = sm.render(line, True, (255, 255, 255))
                s_sh   = sm.render(line, True, (0, 0, 0))
                x = WIDTH - s_surf.get_width() - 6
                y = 6 + k * 14
                screen.blit(s_sh,   (x + 1, y + 1))
                screen.blit(s_surf, (x,     y))

            pygame.display.flip()
            clock.tick()
    finally:
        pygame.quit()


def main():
    ap = argparse.ArgumentParser(
        description="FPGA raytracer sandbox - two-flightstick scene puppet"
    )
    ap.add_argument("host")
    ap.add_argument("port", type=int)
    args = ap.parse_args()

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((args.host, args.port))
    try:
        run(sock)
    finally:
        sock.close()


if __name__ == "__main__":
    main()
```

4 Python Reference + Cocotb Tests

raytracer_fp.py

```
"""
Basic ray tracer: sphere on a plane with an orbiting light source.
Fixed-point version for embedded systems.
Uses Q14.13 format: 14-bit integer + 13-bit fractional (signed, total 27 bits).
"""

from PIL import Image
import math
```

```

class FixedPoint:
    SHIFT = 13
    SCALE = 1 << SHIFT

    @staticmethod
    def from_float(f):
        return int(round(f * FixedPoint.SCALE))

    @staticmethod
    def to_float(fx):
        return fx / FixedPoint.SCALE

    @staticmethod
    def mul(a, b):
        return (a * b) >> FixedPoint.SHIFT

    @staticmethod
    def div(a, b):
        return (a << FixedPoint.SHIFT) // b if b != 0 else 0

    @staticmethod
    def sqrt(fx):
        if fx <= 0:
            return 0
        # Newton's method for fixed-point sqrt
        x = fx
        for _ in range(8):
            x = (x + FixedPoint.div(fx, x)) >> 1
        return x

    @staticmethod
    def sin_approx(fx):
        # Map to [-pi, pi] range
        two_pi = FixedPoint.from_float(2 * math.pi)
        fx = fx % two_pi
        if fx > FixedPoint.from_float(math.pi):
            fx = fx - two_pi

        # Polynomial approximation for sin(x)
        x2 = FixedPoint.mul(fx, fx)
        c1 = FixedPoint.from_float(1.0)
        c3 = FixedPoint.from_float(-1.0/6.0)
        c5 = FixedPoint.from_float(1.0/120.0)
        c7 = FixedPoint.from_float(-1.0/5040.0)

        term1 = FixedPoint.mul(x2, c3)
        term2 = FixedPoint.mul(FixedPoint.mul(x2, x2), c5)
        term3 = FixedPoint.mul(FixedPoint.mul(FixedPoint.mul(x2, x2), x2), c7)

        return fx + FixedPoint.mul(fx, term1 + term2 + term3)

    @staticmethod
    def cos_approx(fx):
        # cos(x) = sin(x + pi/2)
        pi_2 = FixedPoint.from_float(math.pi / 2)
        return FixedPoint.sin_approx(fx + pi_2)

# Fixed-point 3D vector
class Vec3FX:
    def __init__(self, x, y, z):
        self.x = x if isinstance(x, int) else FixedPoint.from_float(x)
        self.y = y if isinstance(y, int) else FixedPoint.from_float(y)
        self.z = z if isinstance(z, int) else FixedPoint.from_float(z)

    def __add__(self, other):
        return Vec3FX(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return Vec3FX(self.x - other.x, self.y - other.y, self.z - other.z)

    def __mul__(self, scalar):
        if isinstance(scalar, Vec3FX):
            return Vec3FX(
```

```
                FixedPoint.mul(self.x, scalar.x),
                FixedPoint.mul(self.y, scalar.y),
                FixedPoint.mul(self.z, scalar.z)
            )
        return Vec3FX(
            FixedPoint.mul(self.x, scalar),
            FixedPoint.mul(self.y, scalar),
            FixedPoint.mul(self.z, scalar)
        )

    def __rmul__(self, scalar):
        return self * scalar

    def dot(self, other):
        return (FixedPoint.mul(self.x, other.x) +
                FixedPoint.mul(self.y, other.y) +
                FixedPoint.mul(self.z, other.z))

    def cross(self, other):
        return Vec3FX(
            FixedPoint.mul(self.y, other.z) - FixedPoint.mul(self.z, other.y),
            FixedPoint.mul(self.z, other.x) - FixedPoint.mul(self.x, other.z),
            FixedPoint.mul(self.x, other.y) - FixedPoint.mul(self.y, other.x)
        )

    def norm_sq(self):
        return self.dot(self)

    def norm(self):
        return FixedPoint.sqrt(self.norm_sq())

    def normalize(self):
        n = self.norm()
        if n > 0:
            inv_n = FixedPoint.div(FixedPoint.SCALE, n)
            return self * inv_n
        return self

    def to_float_array(self):
        return [FixedPoint.to_float(self.x),
                FixedPoint.to_float(self.y),
                FixedPoint.to_float(self.z)]

    def copy(self):
        return Vec3FX(self.x, self.y, self.z)


# --- Ray-object intersection ---
def intersect_sphere(origin, direction, center, radius):
    # Mirrors HW (raytracer.sv stage C/D): half-discriminant form. Python's
    # earlier full-b form was algebraically equivalent but rounded its
    # extra multiplies differently, so at higher precision (Q14.13+) some
    # sphere-tangent pixels disagreed with HW on hit/miss.
    oc = origin - center
    a = direction.dot(direction)
    half_b = direction.dot(oc)
    c = oc.dot(oc) - FixedPoint.mul(radius, radius)
    disc = FixedPoint.mul(half_b, half_b) - FixedPoint.mul(a, c)
    if disc < 0:
        return None
    sqrt_disc = FixedPoint.sqrt(disc)
    t0 = FixedPoint.div(-half_b - sqrt_disc, a)
    t1 = FixedPoint.div(-half_b + sqrt_disc, a)
    t = t0 if t0 > 0 else t1
    return t if t > 0 else None

def intersect_plane(origin, direction, point, normal):
    denom = normal.dot(direction)
    if abs(denom) < FixedPoint.from_float(1e-6):
        return None
    t = FixedPoint.div((point - origin).dot(normal), denom)
    return t if t > 0 else None

def intersect_cone(origin, direction, apex, k_sq, height):
```

```
    # Y-axis cone, apex on top (at apex), opens downward over y
    # [apex.y - height, apex.y]. Surface (-xAx)² + (-zAz)² = k²·(-yAy)².
    # Mirrors the HW (raytracer.sv stage C/D): quadratic at²+2bt+c=0 in
    # the same half-discriminant form as sphere, plus a flat disc cap at
    # y = apex.y - height with radius² = k²·height². Returns (t, normal)
    # for the closest forward hit on either side or cap, or (None, None).
    delta = origin - apex
    a = (FixedPoint.mul(direction.x, direction.x) +
         FixedPoint.mul(direction.z, direction.z) -
         FixedPoint.mul(k_sq, FixedPoint.mul(direction.y, direction.y)))
    half_b = (FixedPoint.mul(delta.x, direction.x) +
              FixedPoint.mul(delta.z, direction.z) -
              FixedPoint.mul(k_sq, FixedPoint.mul(delta.y, direction.y)))
    c = (FixedPoint.mul(delta.x, delta.x) +
         FixedPoint.mul(delta.z, delta.z) -
         FixedPoint.mul(k_sq, FixedPoint.mul(delta.y, delta.y)))

    t_side = None
    if a != 0:
        disc = FixedPoint.mul(half_b, half_b) - FixedPoint.mul(a, c)
        if disc >= 0:
            sqrt_disc = FixedPoint.sqrt(disc)
            t0 = FixedPoint.div(-half_b - sqrt_disc, a)
            t1 = FixedPoint.div(-half_b + sqrt_disc, a)
            # try both roots; accept the smallest forward-hit whose hit_y
            # falls within the single-half slab
            y_top = apex.y
            y_bot = apex.y - height
            for cand in sorted([t0, t1]):
                if cand <= 0:
                    continue
                hit_y = origin.y + FixedPoint.mul(cand, direction.y)
                if y_bot <= hit_y <= y_top:
                    t_side = cand
                    break

    # Bottom disc cap: plane at y = apex.y - height with normal (0,-1,0).
    t_cap = None
    if direction.y != 0:
        cand = FixedPoint.div(apex.y - height - origin.y, direction.y)
        if cand > 0:
            hit_x = origin.x + FixedPoint.mul(cand, direction.x)
            hit_z = origin.z + FixedPoint.mul(cand, direction.z)
            dx = hit_x - apex.x
            dz = hit_z - apex.z
            r_sq_hit = FixedPoint.mul(dx, dx) + FixedPoint.mul(dz, dz)
            r_base_sq = FixedPoint.mul(k_sq, FixedPoint.mul(height, height))
            if r_sq_hit <= r_base_sq:
                t_cap = cand

    candidates = [(t, kind) for t, kind in [(t_side, 'side'), (t_cap, 'cap')]
                  if t is not None]
    if not candidates:
        return None, None
    t, kind = min(candidates, key=lambda x: x[0])

    if kind == 'side':
        # Outward gradient of (-xAx)² + (-zAz)² - k²(-yAy)² = 0.
        hit = origin + direction * t
        n = Vec3FX(hit.x - apex.x,
                   -FixedPoint.mul(k_sq, hit.y - apex.y),
                   hit.z - apex.z)
        return t, n.normalize()
    else:
        return t, Vec3FX(0.0, -1.0, 0.0)

# --- Scene definition ---
# Two-sphere scene matching the HW pipeline (raytracer_batch_mm defaults
# and the cocotb test). Sphere 0 = blue at original position; sphere 1 =
# red, 2.5 units to the +x of sphere 0.
SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
SPHERE0_RADIUS = FixedPoint.from_float(1.0)
SPHERE0_COLOR  = Vec3FX(0.2, 0.3, 0.8)

```

```
SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
SPHERE1_RADIUS = FixedPoint.from_float(1.0)
SPHERE1_COLOR  = Vec3FX(0.8, 0.3, 0.2)

# Back-compat aliases - kept so test_raytracer_batch.py's `from raytracer_fp
# import ...` still works for code that hasn't migrated to the two-sphere
# tuple form.
SPHERE_CENTER = SPHERE0_CENTER
SPHERE_RADIUS = SPHERE0_RADIUS
SPHERE_COLOR  = SPHERE0_COLOR

PLANE_POINT = Vec3FX(0.0, 0.0, 0.0)
PLANE_NORMAL = Vec3FX(0.0, 1.0, 0.0)

PLANE_COLOR1 = Vec3FX(0.9, 0.9, 0.9)
PLANE_COLOR2 = Vec3FX(0.4, 0.4, 0.4)

# Y-axis cone, apex on top, opens downward. Defaults match the HW
# defaults in raytracer_batch_mm: apex at origin, k²=1.0 (45° half-angle),
# height=2.0m, mid-grey, matte (reflect_shift=0). CONE_ENABLED gates the
# cone out of the scene when False - set to False for sphere-only tests.
CONE_ENABLED   = False
CONE_APEX      = Vec3FX(0.0, 2.0, 4.0)
CONE_K_SQ      = FixedPoint.from_float(1.0)
CONE_HEIGHT    = FixedPoint.from_float(2.0)
CONE_COLOR     = Vec3FX(0.5, 0.5, 0.5)
CONE_REFLECT_SHIFT = 0  # 0 = matte;  1 means (color × bright) >> shift, like spheres

AMBIENT = FixedPoint.from_float(0.15)
SPECULAR_EXP = 64
SPECULAR_STRENGTH = FixedPoint.from_float(0.6)

WIDTH, HEIGHT = 480, 360
ASPECT = WIDTH / HEIGHT

# Camera with a vertical image plane: same y for origin/target makes
# forward = (0, 0, 1), right = (-1, 0, 0), up = (0, 1, 0).
CAM_ORIGIN = Vec3FX(0.0, 1.5, -8.0)
CAM_TARGET = Vec3FX(0.0, 1.5,  0.0)

# Image-plane half-extents at distance 1 from the camera. Chosen directly
# (no tan/FOV), so both the Python reference and the SV constants are exact.
# HALF_H = HALF_W / ASPECT = 0.5 * 3/4.
HALF_W = 0.5
HALF_H = 0.375

# --- Camera basis ---
def camera_basis(origin, target):
    forward = (target - origin).normalize()
    y_axis = Vec3FX(0.0, 1.0, 0.0)
    right = forward.cross(y_axis).normalize()
    up = right.cross(forward)
    return forward, right, up

# --- Checkerboard pattern for the plane ---
def plane_color(hit):
    scale = 1
    cx = int(FixedPoint.to_float(hit.x) * scale)
    cz = int(FixedPoint.to_float(hit.z) * scale)
    if (cx + cz) % 2 == 0:
        return PLANE_COLOR1
    return PLANE_COLOR2

# --- Trace a single ray ---
def trace(origin, direction, light_pos):
    """Closest-t intersection across {sphere0, sphere1, plane, cone} then
    shade. Mirrors the HW pipeline: raw (un-normalized) direction,
    hit_type-driven self-skip on shadow rays, HW's >>FRAC_BITS shadow
    offset. Cone is included only if CONE_ENABLED is True."""
    t_s0 = intersect_sphere(origin, direction, SPHERE0_CENTER, SPHERE0_RADIUS)
    t_s1 = intersect_sphere(origin, direction, SPHERE1_CENTER, SPHERE1_RADIUS)
    t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)
    if CONE_ENABLED:
        t_cn, cone_normal = intersect_cone(origin, direction,
```

```
                                        CONE_APEX, CONE_K_SQ, CONE_HEIGHT)
    else:
        t_cn, cone_normal = None, None

    hit_t = None
    hit_type = None
    if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
        hit_t = t_s0; hit_type = 'sphere0'
    if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
        hit_t = t_s1; hit_type = 'sphere1'
    if t_pl is not None and (hit_t is None or t_pl < hit_t):
        hit_t = t_pl; hit_type = 'plane'
    if t_cn is not None and (hit_t is None or t_cn < hit_t):
        hit_t = t_cn; hit_type = 'cone'

    if hit_type is None:
        # Sky gradient
        t_sky = FixedPoint.div(direction.y + FixedPoint.SCALE, FixedPoint.from_float(2))
        inv_t = FixedPoint.SCALE - t_sky
        sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
        sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
        return sky1 + sky2

    hit = origin + direction * hit_t

    if hit_type == 'sphere0':
        normal = (hit - SPHERE0_CENTER).normalize()
        color = SPHERE0_COLOR
    elif hit_type == 'sphere1':
        normal = (hit - SPHERE1_CENTER).normalize()
        color = SPHERE1_COLOR
    elif hit_type == 'cone':
        normal = cone_normal
        color = CONE_COLOR
    else:
        normal = PLANE_NORMAL.copy()
        color = plane_color(hit)

    # Lighting
    to_light = light_pos - hit
    light_dist = to_light.norm()
    inv_light_dist = FixedPoint.div(FixedPoint.SCALE, light_dist) if light_dist > 0 else 0
    light_dir = to_light * inv_light_dist

    # Shadow check against spheres + cone, with self-skip via hit_type.
    # Shadow origin uses HW's >>FRAC_BITS offset (= norm/1024 in Q10).
    in_shadow = False
    sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
                hit.y + (normal.y >> FixedPoint.SHIFT),
                hit.z + (normal.z >> FixedPoint.SHIFT))
    if hit_type != 'sphere0':
        ts = intersect_sphere(sh, light_dir, SPHERE0_CENTER, SPHERE0_RADIUS)
        if ts is not None and ts < light_dist:
            in_shadow = True
    if not in_shadow and hit_type != 'sphere1':
        ts = intersect_sphere(sh, light_dir, SPHERE1_CENTER, SPHERE1_RADIUS)
        if ts is not None and ts < light_dist:
            in_shadow = True
    if not in_shadow and CONE_ENABLED and hit_type != 'cone':
        ts, _ = intersect_cone(sh, light_dir, CONE_APEX, CONE_K_SQ, CONE_HEIGHT)
        if ts is not None and ts < light_dist:
            in_shadow = True

    if in_shadow:
        return color * AMBIENT

    # Diffuse
    dot_prod = normal.dot(light_dir)
    diff = dot_prod if dot_prod > 0 else 0

    ambient_term = color * AMBIENT
    diffuse_term = color * diff
    result = ambient_term + diffuse_term

```

```
    # Clamp to [0, SCALE]
    result.x = max(0, min(FixedPoint.SCALE, result.x))
    result.y = max(0, min(FixedPoint.SCALE, result.y))
    result.z = max(0, min(FixedPoint.SCALE, result.z))

    return result

# --- Render one frame ---
def render_frame(light_pos):
    forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)

    img = []

    for y in range(HEIGHT):
        row = []
        py_flt = (1.0 - 2.0 * y / HEIGHT) * HALF_H
        py = FixedPoint.from_float(py_flt)
        for x in range(WIDTH):
            px_flt = (2.0 * x / WIDTH - 1.0) * HALF_W
            px = FixedPoint.from_float(px_flt)

            direction = (forward +
                         right * px +
                         up * py)

            result = trace(CAM_ORIGIN, direction, light_pos)

            # Convert fixed-point to 8-bit RGB
            r = int(max(0, min(255, FixedPoint.to_float(result.x) * 255)))
            g = int(max(0, min(255, FixedPoint.to_float(result.y) * 255)))
            b = int(max(0, min(255, FixedPoint.to_float(result.z) * 255)))
            row.append((r, g, b))
        img.append(row)

    return img

# --- Animate ---
def main():
    num_frames = 10
    light_radius_float = 4.0
    light_height = FixedPoint.from_float(4.0)
    frames = []

    for i in range(num_frames):
        angle = 2 * math.pi * i / num_frames
        cos_angle = FixedPoint.cos_approx(FixedPoint.from_float(angle))
        sin_angle = FixedPoint.sin_approx(FixedPoint.from_float(angle))
        light_radius_fx = FixedPoint.from_float(light_radius_float)

        light_pos = Vec3FX(
            FixedPoint.mul(light_radius_fx, cos_angle),
            FixedPoint.to_float(light_height),
            FixedPoint.mul(light_radius_fx, sin_angle),
        )

        print(f"Rendering frame {i + 1}/{num_frames}...")
        pixels = render_frame(light_pos)

        # Convert pixel list to PIL Image
        img_array = [[pixels[y][x] for x in range(WIDTH)] for y in range(HEIGHT)]
        pil_img = Image.new('RGB', (WIDTH, HEIGHT))
        pil_img.putdata([pixel for row in img_array for pixel in row])
        frames.append(pil_img)

    out_path = "raytracer.gif"
    frames[0].save(
        out_path,
        save_all=True,
        append_images=frames[1:],
        duration=80,
        loop=0,
    )
    print(f"Saved animation to {out_path}")

```

```
if __name__ == "__main__":
    main()
```

raytracer_reflect_ref.py

```
"""HW-matching Python reference for the reflection-enabled raytracer.

Replicates `raytracer.sv`'s rtl_batch_pipe + spawn router + accumulator at
the bit level so the output should be byte-identical to the FPGA build,
modulo the same ~12-LSB tolerance the cocotb suite already accepts on
sphere-edge pixels (HW Q12 vs Python-fp drift through the sqrt/div
chain).

Reflection model = true geometric reflection: r = d - 2*(d·n)*n. d is
the parent's raw (un-normalized) ray direction; n is the unit surface
normal. |r| = |d|, so the next bounce's stage C absorbs |r|^2 into
a = |raw|^2 exactly like a primary ray. Origin = parent's hit point,
depth incremented, excluded = parent's hit_type so stage D's no_hit mask
blocks the matching sphere on re-entry.

Color accumulation = sum_{i in chain}(to_byte(stage_K_color_i) >>
total_shift_i), saturated at 0xff per channel - matching the HW
accumulator's (acc + (drop_byte_d1 >> meta.total_shift)) RMW + sat_byte.

Run as a script: renders one frame at the cocotb test's default scene/
light and saves to ./raytracer_reflect.png. Tweak DEFAULTS at the
bottom to match the HW CSR programming you want to compare against.
"""

import math
import sys
from PIL import Image

from raytracer_fp import (
    FixedPoint, Vec3FX, camera_basis,
    intersect_sphere, intersect_plane,
    PLANE_POINT, PLANE_NORMAL,
    PLANE_COLOR1, PLANE_COLOR2, AMBIENT,
    CAM_ORIGIN, CAM_TARGET, WIDTH, HEIGHT, HALF_W, HALF_H,
)

# ---- Scene constants (must match HW localparams in raytracer.sv) ------
# Sphere positions/colors are the cocotb test's DEFAULT_SCENE - same as
# raytracer_batch_mm's defaults so a fresh boot of the FPGA renders the
# same scene this script does. Sphere 1's position is mutable (matches
# the HW SC1_*_NEXT CSRs); set_sphere1_position() can be called before
# rendering to relocate it for, e.g., the demo.py close-pair test.
SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
SPHERE0_RADIUS = FixedPoint.from_float(1.0)
SPHERE0_COLOR  = Vec3FX(0.2, 0.3, 0.8)

SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
SPHERE1_RADIUS = FixedPoint.from_float(1.0)
SPHERE1_COLOR  = Vec3FX(0.8, 0.3, 0.2)


def set_sphere1_position(x: float, y: float, z: float):
    """Move sphere 1's center. Mirrors the HW SC1_X_NEXT/Y/Z CSRs which
    are CSR-driven; sphere 0 stays hardcoded at (0, 1, 0)."""
    global SPHERE1_CENTER
    SPHERE1_CENTER = Vec3FX(x, y, z)

ONE = FixedPoint.SCALE         # 4096 in Q12

# Hit-type encoding mirrors stage D's 2-bit field.
HIT_SKY     = 0b00
HIT_SPHERE0 = 0b01
HIT_SPHERE1 = 0b10
HIT_PLANE   = 0b11

# Sun-billboard color: rtl_hit_resolve overrides the sky gradient with
# this when the ray direction aligns with the light to within cos²
```

```
# 63/64  ( 7.2° half-angle). Constants match raytracer.sv's localparams
# bit-for-bit (Verilog integer division truncates, so (95*ONE)/100 gives
# 7782 in Q14.13, not the rounded 7782.4).
SUN_COL = Vec3FX(ONE, (95 * ONE) // 100, (55 * ONE) // 100)


def _sun_active(direction, light_pos):
    """Mirror rtl_hit_resolve's sun-billboard test (capture at t=15):
    sun_active = (sun_dot  0)  (sun_dot²  K_const · a)
    where sun_dot = direction · light_pos, a = |direction|², and
    K_const = |light_pos|² · 63/64. HW treats light_pos as a direction-
    from-origin (no parallax correction at the hit point), so this uses
    light_pos directly rather than (light_pos - origin)."""
    sun_dot = direction.dot(light_pos)
    if sun_dot < 0:
        return False
    lx_sq = FixedPoint.mul(light_pos.x, light_pos.x)
    ly_sq = FixedPoint.mul(light_pos.y, light_pos.y)
    lz_sq = FixedPoint.mul(light_pos.z, light_pos.z)
    light_sq = lx_sq + ly_sq + lz_sq
    K_const = light_sq - (light_sq >> 6)  # |L|² · 63/64
    a = (FixedPoint.mul(direction.x, direction.x) +
         FixedPoint.mul(direction.y, direction.y) +
         FixedPoint.mul(direction.z, direction.z))
    sun_dot_sq = FixedPoint.mul(sun_dot, sun_dot)
    return sun_dot_sq >= FixedPoint.mul(K_const, a)


# ---- HW-matching helpers ------------------------------------------

def to_byte(v_fp):
    """Mirror to_byte() in raytracer.sv: saturate v >= ONE to 0xff,
    otherwise return the 8 bits immediately below the integer bit
    (= map [0, ONE) to [0, 256)).

    HW guarantees v is clamped to [0, ONE] by stage K's clamp01, so the
    saturate branch only fires at exactly 1.0."""
    if v_fp >= ONE:
        return 0xff
    if v_fp < 0:
        return 0
    return (v_fp >> (FixedPoint.SHIFT - 8)) & 0xff


def sat_byte(v_16):
    """Mirror sat_byte() in raytracer.sv: 16-bit accumulator value
    clamped to 8-bit. high-byte non-zero -> 0xff."""
    if v_16 > 0xff:
        return 0xff
    if v_16 < 0:
        return 0
    return v_16 & 0xff


def shaded_color(origin, direction, light_pos, excluded):
    """Mirror stage K's output: closest-t hit -> shaded color in
    fixed-point [0, ONE], plus the auxiliary data the spawn router
    needs (hit_type, hit point, surface normal).

    Matches the HW exactly:
    - direction is raw (un-normalized); stages C/D absorb |raw| in t.
    - sphere intersect uses the half-discriminant form.
    - meta.excluded ORs into the matching sphere's no_hit (mirrors
      stage D's mask in M2c).
    - plane numer = -origin.y (mirrors stage D's variable plane numer).
    - shadow ray origin = hit + (normal >> FRAC_BITS), per-component
      (mirrors stage I's shadow offset).
    - shadow self-skip: stage I ORs (hit_type == matching sphere) into
      shadow_no_hit. Same here.
    - shading: sky branch passes col through; in_shadow -> col * AMBIENT;
      else col*AMBIENT + col*max(0, dot(norm, ldir)). All clamped to
      [0, ONE].

    Returns (color_vec, hit_type, hit_point, surface_normal). hit_type
```

```
    is HIT_* enum. hit_point/normal are None for sky.
    """
    skip_s0 = (excluded == HIT_SPHERE0)
    skip_s1 = (excluded == HIT_SPHERE1)

    t_s0 = None if skip_s0 else intersect_sphere(
        origin, direction, SPHERE0_CENTER, SPHERE0_RADIUS)
    t_s1 = None if skip_s1 else intersect_sphere(
        origin, direction, SPHERE1_CENTER, SPHERE1_RADIUS)
    # Plane denom uses direction.y; numer = dot(plane_origin - origin, n)
    # = -origin.y. intersect_plane in raytracer_fp.py already takes a
    # plane point + normal, so passing PLANE_POINT and PLANE_NORMAL is
    # equivalent (the math reduces to t = -origin.y / dir.y).
    t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)

    hit_t = None
    hit_type = HIT_SKY
    if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
        hit_t = t_s0; hit_type = HIT_SPHERE0
    if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
        hit_t = t_s1; hit_type = HIT_SPHERE1
    if t_pl is not None and (hit_t is None or t_pl < hit_t):
        hit_t = t_pl; hit_type = HIT_PLANE

    # Sky branch: gradient based on direction.y (HW does the same in
    # stage E's hit_type==00 path). The sun-billboard override at t=16
    # replaces the gradient with SUN_COL when the ray points within
    # ~7.2° of the light direction.
    if hit_type == HIT_SKY:
        if _sun_active(direction, light_pos):
            return SUN_COL.copy(), hit_type, None, None
        t_sky = FixedPoint.div(direction.y + ONE, FixedPoint.from_float(2))
        inv_t = ONE - t_sky
        sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
        sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
        col = sky1 + sky2
        # Sky branch in stage K passes col through clamp01 (no-op since
        # sky gradient is already in [0, ONE]).
        col.x = max(0, min(ONE, col.x))
        col.y = max(0, min(ONE, col.y))
        col.z = max(0, min(ONE, col.z))
        return col, hit_type, None, None

    # Hit point + base color + normal.
    hit = origin + direction * hit_t
    if hit_type == HIT_SPHERE0:
        normal = (hit - SPHERE0_CENTER).normalize()
        color = SPHERE0_COLOR
    elif hit_type == HIT_SPHERE1:
        normal = (hit - SPHERE1_CENTER).normalize()
        color = SPHERE1_COLOR
    else:  # plane
        normal = PLANE_NORMAL.copy()
        cx = int(FixedPoint.to_float(hit.x))
        cz = int(FixedPoint.to_float(hit.z))
        color = PLANE_COLOR1 if (cx + cz) % 2 == 0 else PLANE_COLOR2

    # Light direction (normalized).
    to_light = light_pos - hit
    light_dist = to_light.norm()
    inv_ld = FixedPoint.div(ONE, light_dist) if light_dist > 0 else 0
    light_dir = to_light * inv_ld

    # Shadow: HW's hit_type-driven self-skip + per-component norm/1024
    # offset for the shadow origin.
    sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
                hit.y + (normal.y >> FixedPoint.SHIFT),
                hit.z + (normal.z >> FixedPoint.SHIFT))
    in_shadow = False
    if hit_type != HIT_SPHERE0:
        ts = intersect_sphere(sh, light_dir, SPHERE0_CENTER, SPHERE0_RADIUS)
        if ts is not None and ts < light_dist:
            in_shadow = True
    if not in_shadow and hit_type != HIT_SPHERE1:
```

```
        ts = intersect_sphere(sh, light_dir, SPHERE1_CENTER, SPHERE1_RADIUS)
        if ts is not None and ts < light_dist:
            in_shadow = True

    if in_shadow:
        r = color * AMBIENT
    else:
        dot_prod = normal.dot(light_dir)
        diff = dot_prod if dot_prod > 0 else 0
        r = color * AMBIENT + color * diff

    # clamp01 (matches stage K's tick=14 saturate).
    r.x = max(0, min(ONE, r.x))
    r.y = max(0, min(ONE, r.y))
    r.z = max(0, min(ONE, r.z))
    return r, hit_type, hit, normal


def trace_chain(origin, direction, light_pos,
                max_depth, reflect_shifts,
                depth=0, excluded=HIT_SKY, total_shift=0):
    """Mirror the HW spawn-loop: shade this ray, accumulate weighted
    contribution, and if it hit a sphere AND depth < max_depth, recurse
    with origin=hit, direction=r=d-2*(d·n)*n, excluded=hit_type,
    total_shift+= reflect_shifts[hit_type].

    Returns three 16-bit accumulator-style ints (one per channel) that
    the caller saturates with sat_byte.
    """
    col, hit_type, hit_pos, normal = shaded_color(
        origin, direction, light_pos, excluded)

    # weighted contribution = to_byte(col) >> total_shift; HW also caps
    # the shift at 8 (anything past saturates to 0).
    if total_shift >= 8:
        wr = wg = wb = 0
    else:
        wr = to_byte(col.x) >> total_shift
        wg = to_byte(col.y) >> total_shift
        wb = to_byte(col.z) >> total_shift

    # Spawn check: HW spawn_decision = (depth < max_depth) &&
    # (hit_type == sphere0 || sphere1).
    if depth < max_depth and hit_type in (HIT_SPHERE0, HIT_SPHERE1):
        child_shift = reflect_shifts[0] if hit_type == HIT_SPHERE0 \
                                        else reflect_shifts[1]
        # r = d - 2*(d·n)*n. d is non-unit (raw), n is unit; result is
        # |d|*r - same magnitude as d, so stage C's a=|raw|^2 absorbs it.
        dot_dn  = direction.dot(normal)
        two_dot = dot_dn << 1
        refl_dir = direction - normal * two_dot
        cr, cg, cb = trace_chain(
            hit_pos, refl_dir, light_pos,
            max_depth, reflect_shifts,
            depth=depth + 1,
            excluded=hit_type,
            total_shift=total_shift + child_shift,
        )
        return wr + cr, wg + cg, wb + cb

    return wr, wg, wb


# ---- Per-pixel render ------------------------------------------------

# Mirror the cocotb test's pixel_to_screen exactly so px/py drop into
# the same Q-format coords the HW computes in stage A.
_RECIP_SHIFT = 24
_HALF_W_Q = (1 << FixedPoint.SHIFT) // 2
_HALF_H_Q = (_HALF_W_Q * HEIGHT) // WIDTH
_INV_W_Q = ((2 * _HALF_W_Q) << _RECIP_SHIFT) // WIDTH
_INV_H_Q = ((2 * _HALF_H_Q) << _RECIP_SHIFT) // HEIGHT


```

```
def pixel_to_screen(x, y):
    px_c = ((x * _INV_W_Q) >> _RECIP_SHIFT) - _HALF_W_Q
    py_c = _HALF_H_Q - ((y * _INV_H_Q) >> _RECIP_SHIFT)
    return px_c, py_c


def camera_basis_from_euler(yaw_rad, pitch_rad):
    """Float-domain basis from yaw + pitch, matching the Euler convention
    in SW/desktop/camera.py:camera_basis_from_euler. yaw=0, pitch=0 yields
    right=(-1,0,0), up=(0,1,0), fwd=(0,0,1) - the legacy hardcoded scene.
    Returns three Vec3FX (right.y is hardcoded 0 - no roll)."""
    cy, sy = math.cos(yaw_rad),   math.sin(yaw_rad)
    cp, sp = math.cos(pitch_rad), math.sin(pitch_rad)
    right   = Vec3FX(-cy,               0.0, sy)
    up      = Vec3FX(-sy * sp,          cp,  -cy * sp)
    forward = Vec3FX( sy * cp,          sp,   cy * cp)
    return forward, right, up


def render_pixel(x, y, light_pos, max_depth, reflect_shifts,
                 cam_origin, forward, right, up):
    px_fp, py_fp = pixel_to_screen(x, y)
    raw = forward + right * px_fp + up * py_fp
    ar, ag, ab = trace_chain(cam_origin, raw, light_pos,
                             max_depth, reflect_shifts)
    return sat_byte(ar), sat_byte(ag), sat_byte(ab)


def render_frame(light_pos, max_depth=1, reflect_shifts=(2, 1),
                 cam_origin=None, yaw_rad=0.0, pitch_rad=0.0):
    """Full WIDTH x HEIGHT render. Returns a PIL Image.

    cam_origin defaults to the legacy CAM_ORIGIN = (0, 1.5, -8) when
    None. yaw/pitch default to 0 - identity basis matching the legacy
    look-at-target setup. To match a runtime HW frame, pass cam_origin
    + the same yaw/pitch the HW received via its CSRs."""
    if cam_origin is None:
        cam_origin = CAM_ORIGIN
    forward, right, up = camera_basis_from_euler(yaw_rad, pitch_rad)
    img = Image.new("RGB", (WIDTH, HEIGHT))
    pixels = []
    for y in range(HEIGHT):
        for x in range(WIDTH):
            pixels.append(render_pixel(x, y, light_pos, max_depth,
                                       reflect_shifts, cam_origin,
                                       forward, right, up))
        if y % 30 == 0:
            print(f"  row {y}/{HEIGHT}", file=sys.stderr)
    img.putdata(pixels)
    return img


# ---- Defaults match raytracer_batch_mm's reset state -----------------

DEFAULT_MAX_DEPTH      = 3
DEFAULT_REFLECT_SHIFT_0 = 2  # 25% contribution from sphere 0 reflection
DEFAULT_REFLECT_SHIFT_1 = 1  # 50% from sphere 1
DEFAULT_LIGHT          = Vec3FX(4.0, 4.0, 0.0)


def main():
    """Render one frame at the HW reset defaults + center light."""
    import argparse
    p = argparse.ArgumentParser(
        description="HW-matching reflection raytracer reference."
    )
    p.add_argument("--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
                   help="recursion cap (0=no reflection; default %(default)d)")
    p.add_argument("--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0,
                   help="sphere 0 reflect_shift (default %(default)d)")
    p.add_argument("--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1,
                   help="sphere 1 reflect_shift (default %(default)d)")
    p.add_argument("--light-x", type=float, default=4.0)
    p.add_argument("--light-y", type=float, default=4.0)
```

```
    p.add_argument("--light-z", type=float, default=0.0)
    p.add_argument("--sphere1", nargs=3, type=float, metavar=("X", "Y", "Z"),
                   help="sphere 1 (red) center (default 2.5 1.0 0.0)")
    # Runtime camera (matches the HW CSR-driven camera state). Defaults
    # reproduce the legacy hardcoded scene exactly. cam_y is locked at 1.5
    # in HW so it has no CLI knob.
    p.add_argument("--cam-x", type=float, default=0.0,
                   help="camera origin x (default 0.0)")
    p.add_argument("--cam-z", type=float, default=-8.0,
                   help="camera origin z (default -8.0)")
    p.add_argument("--yaw", type=float, default=0.0,
                   help="camera yaw in degrees (default 0)")
    p.add_argument("--pitch", type=float, default=0.0,
                   help="camera pitch in degrees (default 0)")
    p.add_argument("--out", default="raytracer_reflect.png",
                   help="output path (default %(default)s)")
    p.add_argument("--gif-frames", type=int, default=0,
                   help="if >0, render an orbiting-light GIF with N frames")
    args = p.parse_args()

    if args.sphere1 is not None:
        set_sphere1_position(*args.sphere1)

    light = Vec3FX(args.light_x, args.light_y, args.light_z)
    shifts = (args.shift0, args.shift1)
    # cam_y locked at 1.5 in HW (no Y-translation per camera plan).
    cam_origin = Vec3FX(args.cam_x, 1.5, args.cam_z)
    yaw_rad   = math.radians(args.yaw)
    pitch_rad = math.radians(args.pitch)

    if args.gif_frames > 0:
        light_radius = math.sqrt(args.light_x**2 + args.light_z**2) or 4.0
        frames = []
        for i in range(args.gif_frames):
            angle = 2 * math.pi * i / args.gif_frames
            lp = Vec3FX(
                light_radius * math.cos(angle),
                args.light_y,
                light_radius * math.sin(angle),
            )
            print(f"frame {i+1}/{args.gif_frames} (light angle "
                  f"{math.degrees(angle):.1f}°)", file=sys.stderr)
            frames.append(render_frame(lp, args.max_depth, shifts,
                                       cam_origin=cam_origin,
                                       yaw_rad=yaw_rad,
                                       pitch_rad=pitch_rad))
        out_path = args.out if args.out.endswith(".gif") else \
                   args.out.rsplit(".", 1)[0] + ".gif"
        frames[0].save(out_path, save_all=True,
                       append_images=frames[1:], duration=80, loop=0)
        print(f"saved {out_path}", file=sys.stderr)
    else:
        print(f"rendering {WIDTH}x{HEIGHT} at max_depth={args.max_depth}, "
              f"shifts=({args.shift0},{args.shift1}), "
              f"cam=({args.cam_x:+.2f}, 1.50, {args.cam_z:+.2f}), "
              f"yaw={args.yaw:+.1f}° pitch={args.pitch:+.1f}°",
              file=sys.stderr)
        img = render_frame(light, args.max_depth, shifts,
                           cam_origin=cam_origin,
                           yaw_rad=yaw_rad,
                           pitch_rad=pitch_rad)
        img.save(args.out)
        print(f"saved {args.out}", file=sys.stderr)


if __name__ == "__main__":
    main()
```

test_raytracer_batch.py

```
import cocotb
from cocotb.clock import Clock
```

```
from cocotb.triggers import RisingEdge
import sys, os, math

sys.path.insert(0, os.path.dirname(__file__))
from raytracer_fp import (FixedPoint, Vec3FX, camera_basis,
                          intersect_sphere, intersect_plane,
                          PLANE_POINT, PLANE_NORMAL,
                          PLANE_COLOR1, PLANE_COLOR2, AMBIENT,
                          CAM_ORIGIN, CAM_TARGET, WIDTH, HEIGHT,
                          HALF_W, HALF_H)

WORD = 27
N    = 120          # must match the BATCH parameter in rtl_batch_pipe
TOLERANCE = 12      # 8-bit channel LSBs; covers py-model vs HW-fp drift
                    # on sphere-edge pixels

# Two-sphere scene. Sphere 0 keeps the historical center (matches the old
# 1-sphere golden); sphere 1 sits 2.5 units to the right. Colors picked to
# visually distinguish them. Same defaults as raytracer_batch_mm hardcodes.
SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
SPHERE0_RADIUS = FixedPoint.from_float(1.0)
SPHERE0_COLOR  = Vec3FX(0.2, 0.3, 0.8)

SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
SPHERE1_RADIUS = FixedPoint.from_float(1.0)
SPHERE1_COLOR  = Vec3FX(0.8, 0.3, 0.2)

# Sphere colors are baked into stage E as localparams (SCOL0_*, SCOL1_*) so
# they are NOT runtime-configurable. Tests that change sphere positions/radii
# via the dut still get the same colors back from HW.
def _scene(sc0=SPHERE0_CENTER, r0=SPHERE0_RADIUS,
           sc1=SPHERE1_CENTER, r1=SPHERE1_RADIUS):
    return (sc0, r0, sc1, r1)

DEFAULT_SCENE = _scene()


def trace_py(origin, direction, light_pos, scene=DEFAULT_SCENE):
    """Python reference trace for the two-sphere pipeline.

    Mirrors HW: closest-t winner across {sphere0, sphere1, plane}; raw
    (un-normalized) direction; shadow ray with HW's >>FRAC_BITS offset and
    self-skip on the matching sphere via hit_type."""
    sc0, r0, sc1, r1 = scene
    t_s0 = intersect_sphere(origin, direction, sc0, r0)
    t_s1 = intersect_sphere(origin, direction, sc1, r1)
    t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)

    hit_t = None
    hit_type = None
    if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
        hit_t = t_s0; hit_type = 'sphere0'
    if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
        hit_t = t_s1; hit_type = 'sphere1'
    if t_pl is not None and (hit_t is None or t_pl < hit_t):
        hit_t = t_pl; hit_type = 'plane'

    if hit_type is None:
        SCALE = FixedPoint.SCALE
        t_sky = FixedPoint.div(direction.y + SCALE, FixedPoint.from_float(2))
        inv_t = SCALE - t_sky
        sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
        sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
        r = sky1 + sky2
        return (r.x, r.y, r.z)

    hit = origin + direction * hit_t
    if hit_type == 'sphere0':
        normal = (hit - sc0).normalize()
        color = SPHERE0_COLOR
    elif hit_type == 'sphere1':
        normal = (hit - sc1).normalize()
        color = SPHERE1_COLOR
    else:
```

```
        normal = PLANE_NORMAL.copy()
        cx = int(FixedPoint.to_float(hit.x))
        cz = int(FixedPoint.to_float(hit.z))
        color = PLANE_COLOR1 if (cx + cz) % 2 == 0 else PLANE_COLOR2

    to_light = light_pos - hit
    light_dist = to_light.norm()
    inv_ld = FixedPoint.div(FixedPoint.SCALE, light_dist) if light_dist > 0 else 0
    light_dir = to_light * inv_ld

    # Shadow ray. HW computes shad_origin = hit + (norm >>> FRAC_BITS),
    # which is the per-component arithmetic right shift in fixed point.
    in_shadow = False
    if hit_type != 'sky':
        sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
                    hit.y + (normal.y >> FixedPoint.SHIFT),
                    hit.z + (normal.z >> FixedPoint.SHIFT))
        if hit_type != 'sphere0':
            ts = intersect_sphere(sh, light_dir, sc0, r0)
            if ts is not None and ts < light_dist:
                in_shadow = True
        if not in_shadow and hit_type != 'sphere1':
            ts = intersect_sphere(sh, light_dir, sc1, r1)
            if ts is not None and ts < light_dist:
                in_shadow = True

    if in_shadow:
        r = color * AMBIENT
    else:
        dot_prod = normal.dot(light_dir)
        diff = dot_prod if dot_prod > 0 else 0
        r = color * AMBIENT + color * diff

    r.x = max(0, min(FixedPoint.SCALE, r.x))
    r.y = max(0, min(FixedPoint.SCALE, r.y))
    r.z = max(0, min(FixedPoint.SCALE, r.z))
    return (r.x, r.y, r.z)

def to_unsigned(val):
    if val < 0:
        val = val + (1 << WORD)
    return val

def pack_rgb(color):
    """Pack raytracer_pkg::rgb_t {r, g, b} (MSB first) into a 3·WORD-bit int."""
    return (
        (to_unsigned(color.x) << (2 * WORD)) |
        (to_unsigned(color.y) << (1 * WORD)) |
        (to_unsigned(color.z))
    )

# sphere_t = {x, y, z, r_sq, color: rgb_t, reflect_shift[4:0]}.
# Width = 4*WORD + 3*WORD + 5 = 7*WORD + 5 = 194 bits at WORD=27.
SPHERE_BITS = 7 * WORD + 5

def pack_sphere(sc, r_sq, color=Vec3FX(0.5, 0.5, 0.5), reflect_shift=0):
    val = 0
    val = (val << WORD) | to_unsigned(sc.x)
    val = (val << WORD) | to_unsigned(sc.y)
    val = (val << WORD) | to_unsigned(sc.z)
    val = (val << WORD) | to_unsigned(r_sq)
    val = (val << WORD) | to_unsigned(color.x)
    val = (val << WORD) | to_unsigned(color.y)
    val = (val << WORD) | to_unsigned(color.z)
    val = (val << 5)    | (int(reflect_shift) & 0x1f)
    return val

# cone_t = {apex_x, apex_y, apex_z, k_sq, height, norm_factor, color, reflect_shift[4:0]}.
# Width = 6*WORD + 3*WORD + 5 = 9*WORD + 5 = 248 bits at WORD=27.
CONE_BITS = 9 * WORD + 5

def pack_cone(apex, k_sq, height, norm_factor,
              color=Vec3FX(0.5, 0.5, 0.5), reflect_shift=0):
    val = 0
```

```
    val = (val << WORD) | to_unsigned(apex.x)
    val = (val << WORD) | to_unsigned(apex.y)
    val = (val << WORD) | to_unsigned(apex.z)
    val = (val << WORD) | to_unsigned(k_sq)
    val = (val << WORD) | to_unsigned(height)
    val = (val << WORD) | to_unsigned(norm_factor)
    val = (val << WORD) | to_unsigned(color.x)
    val = (val << WORD) | to_unsigned(color.y)
    val = (val << WORD) | to_unsigned(color.z)
    val = (val << 5)    | (int(reflect_shift) & 0x1f)
    return val

# floor_t = {pc1: rgb_t, pc2: rgb_t, mode[1:0], lim_x, lim_z}.
# Width = 6*WORD + 2 + 2*WORD = 8*WORD + 2 = 218 bits at WORD=27.
FLOOR_BITS = 8 * WORD + 2

def pack_floor(pc1, pc2, mode=0, lim_x=None, lim_z=None):
    if lim_x is None:
        lim_x = (1 << (WORD - 1)) - 1
    if lim_z is None:
        lim_z = (1 << (WORD - 1)) - 1
    val = 0
    for v in [pc1.x, pc1.y, pc1.z, pc2.x, pc2.y, pc2.z]:
        val = (val << WORD) | to_unsigned(v)
    val = (val << 2)    | (int(mode) & 0x3)
    val = (val << WORD) | to_unsigned(lim_x)
    val = (val << WORD) | to_unsigned(lim_z)
    return val

# scene_t = {sphere_t s0, sphere_t s1, cone_t cone, floor_t floor}.
SCENE_BITS = 2 * SPHERE_BITS + CONE_BITS + FLOOR_BITS

def pack_scene(s0_pack, s1_pack, cone_pack, floor_pack):
    val = s0_pack
    val = (val << SPHERE_BITS) | s1_pack
    val = (val << CONE_BITS)   | cone_pack
    val = (val << FLOOR_BITS)  | floor_pack
    return val


class SceneState:
    """Mutable scene mirror that knows how to push itself into dut.scene.
    Defaults match the wrapper's reset block so the cocotb top renders the
    same picture as a hardware boot."""
    def __init__(self):
        # Sphere 0 - blue, 25% reflective.
        self.s0_center = SPHERE0_CENTER
        self.s0_r_sq   = FixedPoint.mul(SPHERE0_RADIUS, SPHERE0_RADIUS)
        self.s0_color  = SPHERE0_COLOR
        self.s0_reflect_shift = 2
        # Sphere 1 - red, 50% reflective.
        self.s1_center = SPHERE1_CENTER
        self.s1_r_sq   = FixedPoint.mul(SPHERE1_RADIUS, SPHERE1_RADIUS)
        self.s1_color  = SPHERE1_COLOR
        self.s1_reflect_shift = 1
        # Cone - disabled (height=0), matte, mid-grey, k²=1 (45°).
        self.cone_apex = Vec3FX(0.0, 0.0, 0.0)
        self.cone_k_sq = FixedPoint.from_float(1.0)
        self.cone_height = FixedPoint.from_float(0.0)
        self.cone_norm_factor = FixedPoint.from_float(math.sqrt(2.0))
        self.cone_color = Vec3FX(0.5, 0.5, 0.5)
        self.cone_reflect_shift = 0
        # Floor - legacy checker.
        self.floor_pc1 = Vec3FX(0.9, 0.9, 0.9)
        self.floor_pc2 = Vec3FX(0.4, 0.4, 0.4)
        self.floor_mode = 0
        self.floor_lim_x = None  # → max-positive in pack_floor
        self.floor_lim_z = None

    def pack(self):
        s0 = pack_sphere(self.s0_center, self.s0_r_sq, self.s0_color, self.s0_reflect_shift)
        s1 = pack_sphere(self.s1_center, self.s1_r_sq, self.s1_color, self.s1_reflect_shift)
        c  = pack_cone(self.cone_apex, self.cone_k_sq, self.cone_height,
                       self.cone_norm_factor, self.cone_color, self.cone_reflect_shift)
```

```
        fl = pack_floor(self.floor_pc1, self.floor_pc2, self.floor_mode,
                        self.floor_lim_x, self.floor_lim_z)
        return pack_scene(s0, s1, c, fl)

    def drive(self, dut):
        dut.scene.value = self.pack()

    def from_scene_tuple(self, scene_tuple):
        """Update sphere geometry from a (sc0, r0, sc1, r1) tuple. Colors and
        reflect_shift unchanged."""
        sc0, r0, sc1, r1 = scene_tuple
        self.s0_center = sc0
        self.s0_r_sq   = FixedPoint.mul(r0, r0)
        self.s1_center = sc1
        self.s1_r_sq   = FixedPoint.mul(r1, r1)
        return self

def rgb8(fp_val):
    return int(max(0, min(255, FixedPoint.to_float(fp_val) * 255)))

# Mirror stage A's INV_W/INV_H multiply-shift exactly so the golden's raw
# values match HW bit-for-bit. Float-based pixel-to-screen drifts by 1 LSB
# at some pixels, which used to be absorbed by stage B's normalize but now
# tips disc through zero at the sphere silhouette.
_RECIP_SHIFT = 24
_HALF_W_Q = (1 << FixedPoint.SHIFT) // 2
_HALF_H_Q = (_HALF_W_Q * HEIGHT) // WIDTH
_INV_W_Q = ((2 * _HALF_W_Q) << _RECIP_SHIFT) // WIDTH
_INV_H_Q = ((2 * _HALF_H_Q) << _RECIP_SHIFT) // HEIGHT

def pixel_to_screen(x, y):
    """Integer pixel coords -> Q-format screen-space px, py."""
    px_c = ((x * _INV_W_Q) >> _RECIP_SHIFT) - _HALF_W_Q
    py_c = _HALF_H_Q - ((y * _INV_H_Q) >> _RECIP_SHIFT)
    return px_c, py_c

def render_pixel_py(px_fp, py_fp, light_pos, scene=DEFAULT_SCENE):
    """Python reference: compute direction from px/py, call trace_py.

    Stage B (camera-ray normalize) was dropped in HW: stages downstream
    operate on un-normalized raw directly. The sphere/plane quadratics
    absorb the |raw| scaling so hit points are invariant, and dropping
    normalize here keeps the Q10 rounding bit-identical at the
    silhouette edge."""
    forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
    raw = forward + right * px_fp + up * py_fp
    return trace_py(CAM_ORIGIN, raw, light_pos, scene)

async def reset(dut):
    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)


def k_const_from_light(light):
    """Replicate the wrapper's K_const compute for cocotb. The wrapper
    runs a 3-phase u_l_sq + sum + (x - x>>>6) on light_*_reg; in
    test we drive rtl_batch_pipe directly so the test has to feed the
    same wire. K_const = (lx² + ly² + lz²) · 63/64, all in Q FRAC_BITS."""
    lx_sq = FixedPoint.mul(light.x, light.x)
    ly_sq = FixedPoint.mul(light.y, light.y)
    lz_sq = FixedPoint.mul(light.z, light.z)
    light_sq = lx_sq + ly_sq + lz_sq
    return light_sq - (light_sq >> 6)


def drive_default_camera(dut):
    """Set the camera CSR-driven inputs to the historical hardcoded scene:
    cam at (0, 1.5, -8), identity basis (right=x̂, up=ŷ, fwd=ẑ). With these
    defaults the new basis math reduces to (-px, py, ONE) - bit-equal to the
    legacy raw_x_c/y_c/z_c, so existing test goldens still pass."""
    forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
```

```
    dut.cam_x.value   = to_unsigned(CAM_ORIGIN.x)
    dut.cam_y.value   = to_unsigned(CAM_ORIGIN.y)
    dut.cam_z.value   = to_unsigned(CAM_ORIGIN.z)
    dut.right_x.value = to_unsigned(right.x)
    dut.right_z.value = to_unsigned(right.z)
    dut.up_x.value    = to_unsigned(up.x)
    dut.up_y.value    = to_unsigned(up.y)
    dut.up_z.value    = to_unsigned(up.z)
    dut.fwd_x.value   = to_unsigned(forward.x)
    dut.fwd_y.value   = to_unsigned(forward.y)
    dut.fwd_z.value   = to_unsigned(forward.z)

async def do_batch_count_emits(dut, base_x, y, light, scene=DEFAULT_SCENE,
                               scene_state=None):
    """Same as do_batch but also returns (total_emits, unique_addrs)
    so reflection tests can detect the failure mode where complete_count
    reaches BATCH but some lanes never wr_en (or some get hit twice).
    """
    if scene_state is None:
        scene_state = SceneState().from_scene_tuple(scene)
    dut.pixel_x.value  = base_x
    dut.pixel_y.value  = y
    dut.light_x.value  = to_unsigned(light.x)
    dut.light_y.value  = to_unsigned(light.y)
    dut.light_z.value  = to_unsigned(light.z)
    if hasattr(dut, 'K_const'):
        dut.K_const.value = to_unsigned(k_const_from_light(light))
    scene_state.drive(dut)
    drive_default_camera(dut)
    dut.start.value    = 1
    await RisingEdge(dut.clk)
    dut.start.value    = 0

    captured = {}
    total_emits = 0
    unique_addrs = set()
    cycles = 0
    while True:
        await RisingEdge(dut.clk)
        if int(dut.wr_en.value) == 1:
            addr = int(dut.wr_addr.value)
            data = int(dut.wr_data.value)
            captured[addr] = ((data >> 16) & 0xff,
                              (data >>  8) & 0xff,
                               data        & 0xff)
            total_emits += 1
            unique_addrs.add(addr)
        if int(dut.done.value) == 1:
            break
        cycles += 1
        if cycles > 5000:
            raise TimeoutError("raytracer_batch did not assert done")

    results = [captured.get(i, (0, 0, 0)) for i in range(N)]
    return results, cycles, total_emits, len(unique_addrs)


async def do_batch(dut, base_x, y, light, scene=DEFAULT_SCENE,
                   scene_state=None):
    """Start a batch at (base_x, y) and capture per-lane RGB by listening on
    the BRAM write port. If `scene_state` is supplied, it's pushed into
    dut.scene; otherwise a default SceneState is built from the scene tuple."""
    if scene_state is None:
        scene_state = SceneState().from_scene_tuple(scene)
    dut.pixel_x.value  = base_x
    dut.pixel_y.value  = y
    dut.light_x.value  = to_unsigned(light.x)
    dut.light_y.value  = to_unsigned(light.y)
    dut.light_z.value  = to_unsigned(light.z)
    if hasattr(dut, 'K_const'):
        dut.K_const.value = to_unsigned(k_const_from_light(light))
    scene_state.drive(dut)
    drive_default_camera(dut)
    dut.start.value    = 1
```

```
    await RisingEdge(dut.clk)
    dut.start.value    = 0

    captured = {}
    cycles = 0
    while True:
        await RisingEdge(dut.clk)
        if int(dut.wr_en.value) == 1:
            addr = int(dut.wr_addr.value)
            data = int(dut.wr_data.value)
            captured[addr] = ((data >> 16) & 0xff,
                              (data >>  8) & 0xff,
                               data        & 0xff)
        if int(dut.done.value) == 1:
            break
        cycles += 1
        if cycles > 5000:
            raise TimeoutError("raytracer_batch did not assert done")

    results = [captured.get(i, (0, 0, 0)) for i in range(N)]
    return results, cycles


@cocotb.test()
async def test_batch_center(dut):
    """One batch near the screen center; check every lane against Python ref."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    results, cycles = await do_batch(dut, base_x, y, light)
    dut._log.info(f"batch at ({base_x},{y}) took {cycles} cycles")

    fail = []
    for i, (r, g, b) in enumerate(results):
        x = base_x + i
        px_fp, py_fp = pixel_to_screen(x, y)
        py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light)
        py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
        ok = (abs(r - py_r8) <= TOLERANCE and
              abs(g - py_g8) <= TOLERANCE and
              abs(b - py_b8) <= TOLERANCE)
        marker = "OK" if ok else "FAIL"
        dut._log.info(
            f"  lane {i:2d} px={x:3d}: hw=({r},{g},{b}) "
            f"py=({py_r8},{py_g8},{py_b8})  [{marker}]"
        )
        if not ok:
            fail.append((i, x, (r, g, b), (py_r8, py_g8, py_b8)))

    assert not fail, f"{len(fail)} lane(s) mismatched: {fail}"


@cocotb.test()
async def test_batch_multiple_columns(dut):
    """Several batches at different (x, y); also exercises the right-edge case."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    light = Vec3FX(4.0, 4.0, 0.0)
    test_points = [
        (0,         HEIGHT // 2),     # leftmost
        (WIDTH // 4, HEIGHT // 4),    # upper-left quadrant
        (WIDTH - N,  HEIGHT - 1),     # rightmost (just at the edge)
    ]
    for base_x, y in test_points:
        results, cycles = await do_batch(dut, base_x, y, light)
        for i, (r, g, b) in enumerate(results):
            x = base_x + i
```

```
            px_fp, py_fp = pixel_to_screen(x, y)
            py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light)
            py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
            assert (abs(r - py_r8) <= TOLERANCE and
                    abs(g - py_g8) <= TOLERANCE and
                    abs(b - py_b8) <= TOLERANCE), (
                f"batch ({base_x},{y}) lane {i} px={x}: "
                f"hw=({r},{g},{b}) py=({py_r8},{py_g8},{py_b8})"
            )
        dut._log.info(f"batch at ({base_x},{y}): {cycles} cycles, {N} lanes OK")


@cocotb.test()
async def test_two_sphere_occlusion(dut):
    """Sphere 1 placed *in front of* sphere 0 along the camera axis. Pixels
    near the screen center should hit sphere 1 (red), with sphere 0 (blue)
    fully occluded behind it. Verifies the closest-t winner mux in stage D
    actually picks the closer sphere - not just the first one tested.

    Also asserts the OCCLUSION INVARIANT directly: at least one center lane
    must come back red-dominated (R > B). The vanilla HW vs golden compare
    only catches *consistency* between the two; this extra check catches
    the case where both compute "sphere 0" together due to a missing
    closest-t comparison."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    # Sphere 0 stays at (0, 1, 0) r=1 (the historical scene).
    # Sphere 1 sits 3 units closer to camera at (0, 1, -3) r=1.
    # Camera at (0, 1.5, -8) looks at +z, so sphere 1 (z=-3) is in front
    # of sphere 0 (z=0); both project to the same screen region.
    occ_scene = _scene(
        sc0=Vec3FX(0.0, 1.0,  0.0), r0=FixedPoint.from_float(1.0),
        sc1=Vec3FX(0.0, 1.0, -3.0), r1=FixedPoint.from_float(1.0),
    )

    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    results, cycles = await do_batch(dut, base_x, y, light, scene=occ_scene)
    dut._log.info(f"occlusion batch at ({base_x},{y}) took {cycles} cycles")

    # Compare HW vs Python golden across all lanes.
    fail = []
    for i, (r, g, b) in enumerate(results):
        x = base_x + i
        px_fp, py_fp = pixel_to_screen(x, y)
        py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light, scene=occ_scene)
        py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
        ok = (abs(r - py_r8) <= TOLERANCE and
              abs(g - py_g8) <= TOLERANCE and
              abs(b - py_b8) <= TOLERANCE)
        if not ok:
            fail.append((i, x, (r, g, b), (py_r8, py_g8, py_b8)))
    assert not fail, f"{len(fail)} lane(s) mismatched against golden: {fail[:5]}..."

    # Direct occlusion check: at least one center lane is red-dominated
    # (sphere 1's color), proving the closest-t mux fired.
    center = N // 2
    red_dominant = sum(1 for (r, g, b) in results[center-10:center+10]
                       if r > b + 20)
    assert red_dominant > 0, (
        "no red-dominated lane near screen center - sphere 1 is supposed "
        "to occlude sphere 0 but HW returned blue or non-sphere colors. "
        f"Center 20 lanes: {results[center-10:center+10]}"
    )
    dut._log.info(
        f"occlusion verified: {red_dominant}/20 center lanes red-dominated"
    )


@cocotb.test()
```

```
async def test_render_gif(dut):
    """Render an animated GIF using the parallel raytracer_batch DUT.

    Each row is processed in WIDTH/N batches; lanes within a batch run in
    parallel, so a row takes ~N× fewer DUT cycles than the single-pixel test.
    The width is always the DUT's full WIDTH (so the batches stride cleanly);
    if RENDER_W differs we PIL-resize each finished frame at the end.

    Configure via env vars:
        RENDER_W       output width   (default WIDTH; resized via PIL if smaller)
        RENDER_H       row count      (default 360, sampled from HEIGHT with stride)
        RENDER_FRAMES  frame count    (default 1)
        RENDER_OUT     output path    (default raytracer_batch.gif)
    """
    from PIL import Image

    rw       = int(os.environ.get("RENDER_W", str(WIDTH)))
    rh       = int(os.environ.get("RENDER_H", "360"))
    nframes  = int(os.environ.get("RENDER_FRAMES", "1"))
    out_path = os.environ.get("RENDER_OUT", "raytracer_batch.gif")
    # M2c: optional reflection knobs. Defaults preserve historical
    # behavior (no reflection) so existing renders match byte-for-byte.
    max_depth      = int(os.environ.get("RENDER_MAX_DEPTH", "0"))
    reflect_shift_0 = int(os.environ.get("RENDER_REFLECT_SHIFT_0", "1"))
    reflect_shift_1 = int(os.environ.get("RENDER_REFLECT_SHIFT_1", "1"))
    # Optional sphere 1 position override (close-pair scenes etc).
    sc1_x = float(os.environ.get("RENDER_SC1_X", "2.5"))
    sc1_y = float(os.environ.get("RENDER_SC1_Y", "1.0"))
    sc1_z = float(os.environ.get("RENDER_SC1_Z", "0.0"))
    scene = _scene(sc1=Vec3FX(sc1_x, sc1_y, sc1_z))

    # Optional cone primitive. RENDER_CONE_HEIGHT > 0 enables it; default
    # 0 disables (matches HW reset state).
    cone_apex   = Vec3FX(float(os.environ.get("RENDER_CONE_X", "0.0")),
                         float(os.environ.get("RENDER_CONE_Y", "0.0")),
                         float(os.environ.get("RENDER_CONE_Z", "0.0")))
    cone_k_sq   = FixedPoint.from_float(float(os.environ.get("RENDER_CONE_K_SQ", "1.0")))
    cone_height = FixedPoint.from_float(float(os.environ.get("RENDER_CONE_HEIGHT", "0.0")))
    cone_color  = Vec3FX(float(os.environ.get("RENDER_CONE_R", "0.5")),
                         float(os.environ.get("RENDER_CONE_G", "0.5")),
                         float(os.environ.get("RENDER_CONE_B", "0.5")))
    cone_refl   = int(os.environ.get("RENDER_CONE_REFLECT", "0"))
    # Pre-compute the normal-magnitude factor sqrt(k²·(1+k²)) once per
    # frame in SW. HW recovers |n| with one mul instead of 3 squarings + sqrt.
    k_sq_f       = float(os.environ.get("RENDER_CONE_K_SQ", "1.0"))
    cone_nf_v    = FixedPoint.from_float(math.sqrt(k_sq_f * (1.0 + k_sq_f)))

    # Build a SceneState carrying the render-frame env overrides; the
    # geometry-only sphere positions still come from `scene` and are
    # patched in via from_scene_tuple. SceneState defaults handle the
    # rest (floor, sphere colors, etc. - all matching the wrapper reset).
    render_scene = SceneState().from_scene_tuple(scene)
    render_scene.s0_reflect_shift = reflect_shift_0
    render_scene.s1_reflect_shift = reflect_shift_1
    render_scene.cone_apex            = cone_apex
    render_scene.cone_k_sq            = cone_k_sq
    render_scene.cone_height          = cone_height
    render_scene.cone_norm_factor     = cone_nf_v
    render_scene.cone_color           = cone_color
    render_scene.cone_reflect_shift   = cone_refl

    assert WIDTH % N == 0, (
        f"WIDTH={WIDTH} must be a multiple of N={N} so each row has whole batches"
    )
    batches_per_row = WIDTH // N

    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    # Drive reflection recursion cap once after reset; do_batch's start
    # latches scene at each batch's start edge.
    dut.max_depth.value = max_depth

```

```
    light_radius = float(os.environ.get("RENDER_LIGHT_RADIUS", "4.0"))
    # Sun elevation = arctan(light_height / light_radius). 1.0 puts it
    # in the upper third of the frame; raise via RENDER_LIGHT_HEIGHT for
    # a higher sun (e.g. 3.0  37°, 4.0  45°).
    light_height = float(os.environ.get("RENDER_LIGHT_HEIGHT", "1.0"))
    # Optional explicit light position override (single frame only). When
    # any of RENDER_LIGHT_{X,Y,Z} is set, all three are required and the
    # orbit math above is bypassed.
    light_xyz_override = (
        os.environ.get("RENDER_LIGHT_X") if os.environ.get("RENDER_LIGHT_X") is not None
        else os.environ.get("RENDER_LIGHT_Y")
    )

    frames = []
    for f in range(nframes):
        if os.environ.get("RENDER_LIGHT_X") is not None:
            light = Vec3FX(
                float(os.environ["RENDER_LIGHT_X"]),
                float(os.environ["RENDER_LIGHT_Y"]),
                float(os.environ["RENDER_LIGHT_Z"]),
            )
        else:
            angle = 2 * math.pi * f / nframes
            light = Vec3FX(
                light_radius * math.cos(angle),
                light_height,
                light_radius * math.sin(angle),
            )

        dut._log.info(f"Rendering frame {f+1}/{nframes} ({WIDTH}x{rh}) ...")

        pixels = []   # flat list of (r, g, b) tuples in row-major order
        total_cycles = 0
        for y in range(rh):
            dut_y = (y * HEIGHT) // rh
            row_cycles = 0
            for batch_idx in range(batches_per_row):
                base_x = batch_idx * N
                results, c = await do_batch(
                    dut, base_x, dut_y, light, scene,
                    scene_state=render_scene,
                )
                row_cycles += c
                total_cycles += c
                pixels.extend(results)
            dut._log.info(
                f"  frame {f+1}/{nframes} row {y+1}/{rh} done "
                f"({batches_per_row} batches, {row_cycles} cycles)"
            )
        dut._log.info(
            f"Total cycles: {total_cycles}"
        )

        img = Image.new("RGB", (WIDTH, rh))
        img.putdata(pixels)
        if rw != WIDTH:
            img = img.resize((rw, rh))
        frames.append(img)

    frames[0].save(
        out_path,
        save_all=True,
        append_images=frames[1:],
        duration=80,
        loop=0,
    )
    dut._log.info(f"Saved animation to {out_path}")


# ---------------------------------------------------------------------------
# Reflection (M2c) - verifies the spawn router / per-pipe accumulator combo
# actually re-issues a child ray and adds the attenuated reflection color.
# ---------------------------------------------------------------------------

```

```
@cocotb.test()
async def test_reflection_max_depth_1(dut):
    """One bounce of reflection. max_depth=1, reflect_shift_0=2 (= 25%
    contribution of the reflected color). The reflection direction in HW
    is r = d - 2·(d·n)·n (true geometric reflection, computed in stage K).

    Strategy: render the same center batch twice - once with max_depth=0
    (baseline) and once with max_depth=1. For lanes that hit a sphere,
    the reflection child traverses the pipeline using the parent's hit
    point as origin and the surface normal as direction; its shaded
    color (>> reflect_shift) sums into the lane's accumulator slot. So
    sphere lanes should differ between the two runs (reflection adds
    color), while sky lanes should be unchanged (sky rays don't spawn
    children - hit_type=00 fails the spawn gate). The plane case behaves
    like sky here too (hit_type=11 also fails the gate).
    """
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    # Light positioned to cast a non-shadowed sphere0 hit at the screen
    # center (so the primary's color is bright and reflection is visible
    # in the sum).
    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    # Baseline pass: max_depth=0 with all reflect_shifts zeroed.
    dut.max_depth.value = 0
    baseline_scene = SceneState()
    baseline_scene.s0_reflect_shift = 0
    baseline_scene.s1_reflect_shift = 0
    base_results, base_cyc = await do_batch(dut, base_x, y, light,
                                            scene_state=baseline_scene)
    dut._log.info(f"baseline (max_depth=0) batch took {base_cyc} cycles")

    # Reflection pass: max_depth=1, reflect_shift=2 (= 1/4 contribution).
    dut.max_depth.value = 1
    refl_scene = SceneState()
    refl_scene.s0_reflect_shift = 2
    refl_scene.s1_reflect_shift = 2
    refl_results, refl_cyc = await do_batch(dut, base_x, y, light,
                                            scene_state=refl_scene)
    dut._log.info(f"reflection (max_depth=1) batch took {refl_cyc} cycles")

    # Categorize lanes by what the python ref says they hit.
    sphere_hit_lanes = []
    nonsphere_lanes  = []
    for i in range(N):
        x = base_x + i
        px_fp, py_fp = pixel_to_screen(x, y)
        forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
        raw = forward + right * px_fp + up * py_fp
        sc0, r0, sc1, r1 = DEFAULT_SCENE
        t_s0 = intersect_sphere(CAM_ORIGIN, raw, sc0, r0)
        t_s1 = intersect_sphere(CAM_ORIGIN, raw, sc1, r1)
        if t_s0 is not None or t_s1 is not None:
            sphere_hit_lanes.append(i)
        else:
            nonsphere_lanes.append(i)

    dut._log.info(
        f"classified: {len(sphere_hit_lanes)} sphere lanes, "
        f"{len(nonsphere_lanes)} non-sphere lanes"
    )
    assert len(sphere_hit_lanes) > 0, (
        "test setup error: no sphere hits at the chosen pixel coords"
    )

    # Non-sphere lanes (sky/plane) must NOT change - their hit_type
    # never triggers the spawn router. Identical RGB byte-for-byte.
    nonsphere_mismatches = []
    for i in nonsphere_lanes:
        if base_results[i] != refl_results[i]:
```

```
            nonsphere_mismatches.append((i, base_results[i], refl_results[i]))
    assert not nonsphere_mismatches, (
        f"non-sphere lanes changed under reflection (should pass through):"
        f" {nonsphere_mismatches[:5]}"
    )

    # Sphere lanes must change AT LEAST SOMETIMES. Even when the
    # reflected ray hits sky directly above the sphere top, the sky
    # gradient adds a non-zero contribution.
    sphere_diffs = 0
    for i in sphere_hit_lanes:
        if base_results[i] != refl_results[i]:
            sphere_diffs += 1
    dut._log.info(
        f"reflection deltas: {sphere_diffs}/{len(sphere_hit_lanes)} "
        f"sphere lanes show different color with max_depth=1"
    )
    assert sphere_diffs > 0, (
        "no sphere lane changed under reflection - spawn router or "
        "accumulator weighting is broken (chain never re-issues, or "
        "child color always sums to zero)"
    )

    # Spot-check direction: for a lane that hits sphere0 with normal
    # pointing roughly "up", the reflection ray hits sky. Sky's blue
    # channel is high, so reflection adds blue. Check that at least one
    # such lane has its blue channel rise.
    blue_rises = 0
    for i in sphere_hit_lanes:
        _, _, b_base = base_results[i]
        _, _, b_refl = refl_results[i]
        if b_refl > b_base:
            blue_rises += 1
    dut._log.info(
        f"blue-channel rises (sky-reflection signature): "
        f"{blue_rises}/{len(sphere_hit_lanes)} sphere lanes"
    )
    assert blue_rises > 0, (
        "no sphere lane gained blue under reflection - reflected sky "
        "color isn't being added to the accumulator"
    )


@cocotb.test()
async def test_reflection_completion(dut):
    """Sanity: with reflection on, every lane still completes (one
    wr_en write per lane). Catches the failure mode where the spawn
    FIFO gets stuck or re-issued rays never drop, which would leave
    `done` deasserted forever (caught by do_batch's 5000-cycle timeout
    but worth an explicit check).

    Also asserts the reflection batch takes more cycles than the
    baseline - chain rays each cost ~II=37c per pipe stage they
    occupy.
    """
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    dut.max_depth.value = 0
    baseline_scene = SceneState()
    baseline_scene.s0_reflect_shift = 0
    baseline_scene.s1_reflect_shift = 0
    _, base_cyc = await do_batch(dut, base_x, y, light,
                                 scene_state=baseline_scene)

    dut.max_depth.value = 1
    refl_scene = SceneState()
    refl_scene.s0_reflect_shift = 1
    refl_scene.s1_reflect_shift = 1
```

```
    refl_results, refl_cyc, total_emits, unique_addrs = \
        await do_batch_count_emits(dut, base_x, y, light,
                                   scene_state=refl_scene)

    dut._log.info(
        f"reflection: cycles={refl_cyc}, total_emits={total_emits}, "
        f"unique_addrs={unique_addrs}"
    )

    # `done` fires only when complete_count reaches BATCH = N, so we
    # expect exactly N total emits and N unique addresses. A mismatch
    # exposes either dropped emits (FIFO race) or duplicate emits
    # (pending-bit confusion that fires acc_emit twice for one chain).
    assert total_emits == N, (
        f"reflection batch had {total_emits} emits (expected {N}); "
        f"{'extra' if total_emits > N else 'missing'} emits suggest "
        f"a pending-bit or chain-termination bug"
    )
    assert unique_addrs == N, (
        f"reflection batch wrote {unique_addrs} unique tags ({N} expected); "
        f"some addrs got hit twice - pending-bit didn't keep the "
        f"chain in flight, or a stale ray from a prior batch dropped "
        f"into the new batch's slot"
    )

    # Reflection adds work; should take at least as long as baseline.
    # Allow some slack since not every lane spawns (sky/plane lanes
    # don't), so the increase scales with how many lanes hit spheres.
    assert refl_cyc >= base_cyc, (
        f"reflection batch ({refl_cyc} c) ran faster than baseline "
        f"({base_cyc} c) - that's geometrically impossible; chain "
        f"rays can't be free"
    )
    dut._log.info(
        f"completion verified: base={base_cyc}c, refl={refl_cyc}c, "
        f"emits={total_emits}, unique={unique_addrs}"
    )


@cocotb.test()
async def test_reflection_python_golden(dut):
    """Per-lane bit-exact comparison of HW vs raytracer_reflect_ref.py.
    The Python ref reimplements stage K shading + the spawn router +
    accumulator weighting to mirror the HW byte for byte. Same TOLERANCE
    the baseline tests use (~12 LSBs) for sphere-edge drift through
    the sqrt/div fixed-point chain.
    """
    from raytracer_reflect_ref import render_pixel as render_pixel_refl

    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    max_depth = 1
    shift_0   = 2
    shift_1   = 2

    dut.max_depth.value = max_depth
    golden_scene = SceneState()
    golden_scene.s0_reflect_shift = shift_0
    golden_scene.s1_reflect_shift = shift_1
    results, cycles = await do_batch(dut, base_x, y, light,
                                     scene_state=golden_scene)
    dut._log.info(f"reflection golden batch took {cycles} cycles")

    forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
    fail = []
    for i, (r, g, b) in enumerate(results):
        x = base_x + i
        py_r, py_g, py_b = render_pixel_refl(
```

```
            x, y, light, max_depth, (shift_0, shift_1),
            CAM_ORIGIN, forward, right, up
        )
        ok = (abs(r - py_r) <= TOLERANCE and
              abs(g - py_g) <= TOLERANCE and
              abs(b - py_b) <= TOLERANCE)
        if not ok:
            fail.append((i, x, (r, g, b), (py_r, py_g, py_b)))

    if fail:
        for item in fail[:5]:
            dut._log.info(f"  mismatch lane={item[0]} x={item[1]} "
                          f"hw={item[2]} py={item[3]}")
    assert not fail, f"{len(fail)}/{N} lanes mismatch python ref"
    dut._log.info(f"all {N} lanes match raytracer_reflect_ref within ±{TOLERANCE}")


@cocotb.test()
async def test_reflection_max_depth_2(dut):
    """Two bounces. max_depth=2 means a primary that hits a sphere
    spawns one child, and that child (depth=1, depth<max_depth=2) can
    itself spawn a grandchild if it hits a sphere too. Total chain
    length up to 3 rays per slot.

    The accumulator must handle 3 RMWs per slot before emitting:
    - Primary: pending=1 (spawn), color += primary_byte >> 0
    - Child:   pending=1 (spawn), color += child_byte  >> reflect_shift
    - Grandchild: pending=0, color += gc_byte >> 2*reflect_shift, emit

    Stresses the FIFO depth, the per-tag chain coherence, and the
    is_spawn_d1 / pending logic across multiple loops.
    """
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())
    await reset(dut)

    light  = Vec3FX(4.0, 4.0, 0.0)
    base_x = WIDTH // 2 - N // 2
    y      = HEIGHT // 2

    dut.max_depth.value = 2
    depth2_scene = SceneState()
    depth2_scene.s0_reflect_shift = 1
    depth2_scene.s1_reflect_shift = 1
    refl_results, refl_cyc, total_emits, unique_addrs = \
        await do_batch_count_emits(dut, base_x, y, light,
                                   scene_state=depth2_scene)

    dut._log.info(
        f"max_depth=2: cycles={refl_cyc}, total_emits={total_emits}, "
        f"unique_addrs={unique_addrs}"
    )
    assert total_emits == N, (
        f"max_depth=2 batch had {total_emits} emits (expected {N})"
    )
    assert unique_addrs == N, (
        f"max_depth=2 batch wrote {unique_addrs} unique tags ({N} expected)"
    )
```

test_cone.py

```
"""Unit tests for cone intersection in raytracer_fp.

Self-contained: runs as `python3 test_cone.py`. Prints one line per case.
Exits 0 if all pass, 1 if any fail. No pytest dep.

Cases cover:
  side hit + correct outward normal
  cap hit + (0, -1, 0) normal
  miss: ray points away
  miss: ray would hit only the double-cone's upper half
  miss: ray passes through cap radius but at y above slab
```

```
  cap radius boundary
  k²  1 (steeper cone): hit point + normal
"""
import math
import sys

import raytracer_fp as rt
from raytracer_fp import FixedPoint as FP, Vec3FX, intersect_cone


# Helpers ---------------------------------------------------------------------
def vfx(x, y, z):
    return Vec3FX(float(x), float(y), float(z))


def fx(v):
    return FP.from_float(float(v))


def to_floats(v):
    return [round(FP.to_float(v.x), 4),
            round(FP.to_float(v.y), 4),
            round(FP.to_float(v.z), 4)]


PASSES = []
FAILS = []


def check(name, cond, detail=""):
    if cond:
        PASSES.append(name)
        print(f"  PASS  {name}")
    else:
        FAILS.append((name, detail))
        print(f"  FAIL  {name}  {detail}")


def approx(a, b, tol=0.05):
    """tol is generous - fp_sqrt+fp_div in Q14.12 has limited precision."""
    return abs(a - b) <= tol


# Standard 45° cone at apex (0,2,4), height 2 → spans  y[0,2], r_base=2 ----
APEX = vfx(0, 2, 4)
KSQ_45 = fx(1.0)        # tan²(45°)
HEIGHT_2 = fx(2.0)


# Test cases ------------------------------------------------------------------
def test_side_hit_front():
    # Ray straight at the cone from in front: (0,1,-4) → +z. Hits at z=3,
    # y=1 (mid-cone). Normal is gradient ((0, -1·(1-2), 3-4)) = (0,1,-1)
    # normalized = (0, 0.707, -0.707).
    o = vfx(0, 1, -4)
    d = vfx(0, 0, 1)
    t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
    check("side hit: t exists", t is not None)
    if t is None:
        return
    check("side hit:  t7", approx(FP.to_float(t), 7.0))
    nf = to_floats(n)
    check("side hit: normal  y0.707", approx(nf[1], 0.7071))
    check("side hit: normal  z-0.707", approx(nf[2], -0.7071))
    check("side hit: normal  x0", approx(nf[0], 0.0))


def test_cap_hit_below():
    # Ray from beneath, pointing straight up: hits the cap at y=0 first
    # (cap is at apex.y - height = 0). Hit_point at apex_xz = (0,0,4).
    # Cap normal is (0,-1,0).
    o = vfx(0, -2, 4)
    d = vfx(0, 1, 0)
    t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
```

```
    check("cap hit below: t exists", t is not None)
    if t is None:
        return
    check("cap hit below:  t2", approx(FP.to_float(t), 2.0))
    nf = to_floats(n)
    check("cap hit below: normal == (0,-1,0)",
          approx(nf[0], 0) and approx(nf[1], -1) and approx(nf[2], 0))


def test_miss_pointing_away():
    # Ray pointing away from cone (negative z, cone is at +z).
    o = vfx(0, 1, 0)
    d = vfx(0, 0, -1)
    t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
    check("miss: pointing away", t is None)


def test_miss_above_apex():
    # Ray entirely above apex (y > 2): horizontal ray at y=5 should miss
    # the single-half cone (lower half).
    o = vfx(0, 5, -4)
    d = vfx(0, 0, 1)
    t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
    check("miss: ray above apex slab", t is None)


def test_cap_boundary():
    # Ray straight up through the cap edge: at apex_xz=(0,0,4), r_base=2,
    # so cap edge is at (2,0,4) say. Origin (2, -1, 4), direction (0,1,0)
    # hits cap exactly at radius 2.
    o = vfx(2, -1, 4)
    d = vfx(0, 1, 0)
    t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
    check("cap boundary: t exists", t is not None)
    if t is None:
        return
    # Could hit either cap (at y=0, t=1) or side; cap is closer.
    check("cap boundary:  t1", approx(FP.to_float(t), 1.0, tol=0.1))


def test_cap_outside():
    # Ray straight up at (3, -1, 4) - radius 3 > r_base=2. Should miss the
    # cap. The side might still be hit if the ray geometry allows, but in
    # this case the ray is parallel to the cone axis (y-aligned), so it's
    # the parallel-to-side case. With k=1, |D_xz|² - k²·D.y² = 0 - 1 = -1
    # (a < 0), so the quadratic might still have real roots. Let's check.
    # Actually a is negative, both roots are real. The side intersection
    # could happen above the cap radius - let's see what the test gives.
    # If it hits the side, that's still a valid hit. But the cap should
    # not register since 3 > r_base=2.
    o = vfx(3, -1, 4)
    d = vfx(0, 1, 0)
    t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
    # If t is not None, must be a side hit not a cap hit. That's OK either
    # way - the math is consistent. We just want it not to be the cap.
    if t is not None:
        # Verify hit is on the side: at hit_y, the implicit eq holds
        ho = FP.to_float(o.x) + FP.to_float(t) * FP.to_float(d.x)
        hy = FP.to_float(o.y) + FP.to_float(t) * FP.to_float(d.y)
        hz = FP.to_float(o.z) + FP.to_float(t) * FP.to_float(d.z)
        # cone surface check (k²=1): (x-Ax)² + (z-Az)² = (y-Ay)²
        lhs = (ho - 0)**2 + (hz - 4)**2
        rhs = (hy - 2)**2
        check("cap-outside: hit lies on side surface (not cap)",
              approx(lhs, rhs, tol=0.5),
              detail=f"lhs={lhs:.3f} rhs={rhs:.3f} hit=({ho:.2f},{hy:.2f},{hz:.2f})")
    else:
        check("cap-outside: ray missed entirely (acceptable)", True)


def test_steep_cone():
    # k²=0.25 → tan ()=0.5, half-angle  26.6°. Steeper, narrower cone.
    # r_base = sqrt(0.25)·height = 0.5·2 = 1. So cap is a 1-radius disc.
    # Ray from (0, -1, 4) +y-axis hits cap at y=0, x=0, z=4.
```

```
    ksq_steep = fx(0.25)
    o = vfx(0, -1, 4)
    d = vfx(0, 1, 0)
    t, n = intersect_cone(o, d, APEX, ksq_steep, HEIGHT_2)
    check("steep cone: cap hit exists", t is not None)
    if t is None:
        return
    check("steep cone: cap  t1", approx(FP.to_float(t), 1.0))
    nf = to_floats(n)
    check("steep cone: cap normal == (0,-1,0)",
          approx(nf[1], -1))


def test_apex_origin_cone():
    # Apex at origin (the SW default scene). Cone covers  y[-2,0]. Ray
    # from (0, -1, -2) +z direction hits side. With k=1, side at (y-0)²
    # = x² + z². At y=-1, that's r=1. Ray goes (0,-1,-2)+t(0,0,1). At
    # t=1 hit (0,-1,-1): (-1)² = 0² + (-1)² → 1=1  . So expected t=1.
    apex_o = vfx(0, 0, 0)
    o = vfx(0, -1, -2)
    d = vfx(0, 0, 1)
    t, _ = intersect_cone(o, d, apex_o, fx(1.0), fx(2.0))
    check("apex-at-origin: t exists", t is not None)
    if t is None:
        return
    check("apex-at-origin:  t1", approx(FP.to_float(t), 1.0))


# Run ---------------------------------------------------------------------
print("Running cone unit tests:")
print()
print("[side_hit_front]")
test_side_hit_front()
print("[cap_hit_below]")
test_cap_hit_below()
print("[miss_pointing_away]")
test_miss_pointing_away()
print("[miss_above_apex]")
test_miss_above_apex()
print("[cap_boundary]")
test_cap_boundary()
print("[cap_outside]")
test_cap_outside()
print("[steep_cone]")
test_steep_cone()
print("[apex_origin_cone]")
test_apex_origin_cone()
print()
print(f"Result: {len(PASSES)} pass, {len(FAILS)} fail")
sys.exit(0 if not FAILS else 1)
```

test_fp.py

```
import cocotb
from cocotb.clock import Clock
from cocotb.triggers import RisingEdge, Timer
import sys, os

sys.path.insert(0, os.path.dirname(__file__))
from raytracer_fp import FixedPoint

WORD = 27

def to_unsigned(val):
    """Convert signed Python int to unsigned for cocotb assignment."""
    if val < 0:
        val = val + (1 << WORD)
    return val

def to_signed(val):
    """Convert unsigned cocotb result back to signed Python int."""
    val = int(val) & ((1 << WORD) - 1)
```

```
    if val >= (1 << (WORD - 1)):
        val -= (1 << WORD)
    return val

# ---- fp_mul tests ----

@cocotb.test()
async def test_fp_mul_basic(dut):
    """Test basic multiplication cases."""
    cases = [
        (1.5, 2.0),
        (-3.0, 0.5),
        (0.1, 0.1),
        (0.0, 5.0),
        (-1.0, -1.0),
        (3.0, -2.5),
        (0.001, 1000.0),
    ]
    for a_f, b_f in cases:
        a = FixedPoint.from_float(a_f)
        b = FixedPoint.from_float(b_f)
        expected = FixedPoint.mul(a, b)

        dut.a.value = to_unsigned(a)
        dut.b.value = to_unsigned(b)
        await Timer(1, units="ns")

        result = to_signed(dut.result.value)
        assert result == expected, (
            f"fp_mul({a_f}, {b_f}): got {result}, expected {expected}"
        )
    dut._log.info("All fp_mul basic tests passed")
```

test_fp_div.py

```
import cocotb
from cocotb.clock import Clock
from cocotb.triggers import RisingEdge, Timer
import sys, os

sys.path.insert(0, os.path.dirname(__file__))
from raytracer_fp import FixedPoint

WORD = 27

def to_unsigned(val):
    if val < 0:
        val = val + (1 << WORD)
    return val

def to_signed(val):
    val = int(val) & ((1 << WORD) - 1)
    if val >= (1 << (WORD - 1)):
        val -= (1 << WORD)
    return val

async def do_div(dut, a_fp, b_fp):
    """Drive one division and wait for done. Returns (result, cycles)."""
    dut.a.value = to_unsigned(a_fp)
    dut.b.value = to_unsigned(b_fp)
    dut.start.value = 1
    await RisingEdge(dut.clk)
    dut.start.value = 0

    cycles = 0
    for _ in range(200):
        await RisingEdge(dut.clk)
        cycles += 1
        if dut.done.value == 1:
            return to_signed(dut.result.value), cycles

    raise TimeoutError("fp_div did not assert done")
```

```

@cocotb.test()
async def test_fp_div_basic(dut):
    """Test basic division cases."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)

    cases = [
        (3.0, 2.0),
        (1.0, 4.0),
        (-6.0, 3.0),
        (7.5, -2.5),
        (-1.0, -1.0),
        (0.0, 1.0),
        (100.0, 0.1),
    ]
    for a_f, b_f in cases:
        a = FixedPoint.from_float(a_f)
        b = FixedPoint.from_float(b_f)
        expected = FixedPoint.div(a, b)

        result, cycles = await do_div(dut, a, b)

        diff = abs(result - expected)
        assert diff <= 1, (
            f"fp_div({a_f}, {b_f}): got {result} ({FixedPoint.to_float(result):.6f}), "
            f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
        )
        dut._log.info(f"fp_div({a_f}, {b_f}) = {FixedPoint.to_float(result):.6f} OK  [{cycles} cycles]")

@cocotb.test()
async def test_fp_div_by_zero(dut):
    """Division by zero should return 0 immediately."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)

    a = FixedPoint.from_float(5.0)
    result, cycles = await do_div(dut, a, 0)
    assert result == 0, f"div by zero: got {result}, expected 0"
    dut._log.info(f"Division by zero returns 0 OK  [{cycles} cycles]")
```

test_fp_sqrt.py

```
import cocotb
from cocotb.clock import Clock
from cocotb.triggers import RisingEdge
import sys, os

sys.path.insert(0, os.path.dirname(__file__))
from raytracer_fp import FixedPoint

WORD = 27

def to_unsigned(val):
    if val < 0:
        val = val + (1 << WORD)
    return val

def to_signed(val):
    val = int(val) & ((1 << WORD) - 1)
```

```
    if val >= (1 << (WORD - 1)):
        val -= (1 << WORD)
    return val

async def do_sqrt(dut, a_fp):
    """Drive one sqrt and wait for done. Returns (result, cycles)."""
    dut.a.value = to_unsigned(a_fp)
    dut.start.value = 1
    await RisingEdge(dut.clk)
    dut.start.value = 0

    cycles = 0
    for _ in range(200):
        await RisingEdge(dut.clk)
        cycles += 1
        if dut.done.value == 1:
            return to_signed(dut.result.value), cycles

    raise TimeoutError("fp_sqrt did not assert done")

@cocotb.test()
async def test_fp_sqrt_basic(dut):
    """Test square root against Python reference."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)

    cases = [1.0, 2.0, 4.0, 0.25, 0.5, 9.0, 16.0, 0.01, 100.0, 3.14159]
    for val in cases:
        a = FixedPoint.from_float(val)
        expected = FixedPoint.sqrt(a)

        result, cycles = await do_sqrt(dut, a)

        diff = abs(result - expected)
        tolerance = 2  # allow ±2 LSB since algorithms differ
        assert diff <= tolerance, (
            f"fp_sqrt({val}): got {result} ({FixedPoint.to_float(result):.6f}), "
            f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
        )
        dut._log.info(
            f"fp_sqrt({val}) = {FixedPoint.to_float(result):.6f} "
            f"(expected {FixedPoint.to_float(expected):.6f}, diff={diff})  [{cycles} cycles]"
        )

@cocotb.test()
async def test_fp_sqrt_zero(dut):
    """sqrt(0) should return 0 immediately."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)

    result, cycles = await do_sqrt(dut, 0)
    assert result == 0, f"sqrt(0): got {result}, expected 0"
    dut._log.info(f"sqrt(0) = 0 OK  [{cycles} cycles]")
```

test_fp_inv.py

```
import cocotb
from cocotb.clock import Clock
from cocotb.triggers import RisingEdge
import sys, os
```

```

sys.path.insert(0, os.path.dirname(__file__))
from raytracer_fp import FixedPoint

WORD = 26


def to_unsigned(val):
    if val < 0:
        val = val + (1 << WORD)
    return val


def to_signed(val):
    val = int(val) & ((1 << WORD) - 1)
    if val >= (1 << (WORD - 1)):
        val -= (1 << WORD)
    return val


async def do_inv(dut, b_fp):
    dut.b.value = to_unsigned(b_fp)
    dut.start.value = 1
    await RisingEdge(dut.clk)
    dut.start.value = 0

    cycles = 0
    for _ in range(200):
        await RisingEdge(dut.clk)
        cycles += 1
        if dut.done.value == 1:
            return to_signed(dut.result.value), cycles

    raise TimeoutError("fp_inv did not assert done")


@cocotb.test()
async def test_fp_inv_basic(dut):
    """1/b matches Python reference (= ONE/b via fp_div)."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
    await RisingEdge(dut.clk)

    cases = [2.0, 4.0, -3.0, -2.5, -1.0, 1.0, 0.1, 8.0, 0.5, 100.0]
    one = FixedPoint.from_float(1.0)
    for b_f in cases:
        b = FixedPoint.from_float(b_f)
        expected = FixedPoint.div(one, b)

        result, cycles = await do_inv(dut, b)

        diff = abs(result - expected)
        assert diff <= 1, (
            f"fp_inv({b_f}): got {result} ({FixedPoint.to_float(result):.6f}), "
            f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
        )
        dut._log.info(f"fp_inv({b_f}) = {FixedPoint.to_float(result):.6f} OK  [{cycles} cycles]")


@cocotb.test()
async def test_fp_inv_by_zero(dut):
    """1/0 should return 0 immediately (matches fp_div's b==0 short-circuit)."""
    clock = Clock(dut.clk, 10, units="ns")
    cocotb.start_soon(clock.start())

    dut.rst_n.value = 0
    await RisingEdge(dut.clk)
    await RisingEdge(dut.clk)
    dut.rst_n.value = 1
```

```
    await RisingEdge(dut.clk)

    result, cycles = await do_inv(dut, 0)
    assert result == 0, f"1/0: got {result}, expected 0"
    dut._log.info(f"1/0 returns 0 OK  [{cycles} cycles]")
```

Makefile.raytracer_batch_pipe

```
SIM = verilator
TOPLEVEL_LANG = verilog
VERILOG_SOURCES = $(shell pwd)/raytracer.sv
TOPLEVEL = rtl_batch_pipe
MODULE = test_raytracer_batch
SIM_BUILD = sim_build_raytracer_batch_pipe
EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC
COMPILE_ARGS += -GK=4 -GBATCH=120

include $(shell cocotb-config --makefiles)/Makefile.sim
```

Makefile.div

```
SIM = verilator
TOPLEVEL_LANG = verilog
VERILOG_SOURCES = $(shell pwd)/raytracer.sv
TOPLEVEL = fp_div
MODULE = test_fp_div
SIM_BUILD = sim_build_div
EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC

include $(shell cocotb-config --makefiles)/Makefile.sim
```

Makefile.mul

```
SIM = verilator
TOPLEVEL_LANG = verilog
VERILOG_SOURCES = $(shell pwd)/raytracer.sv
TOPLEVEL = fp_mul
MODULE = test_fp
EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC

include $(shell cocotb-config --makefiles)/Makefile.sim
```

Makefile.sqrt

```
SIM = verilator
TOPLEVEL_LANG = verilog
VERILOG_SOURCES = $(shell pwd)/raytracer.sv
TOPLEVEL = fp_sqrt
MODULE = test_fp_sqrt
SIM_BUILD = sim_build_sqrt
EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC

include $(shell cocotb-config --makefiles)/Makefile.sim
```

Makefile.fp_inv

```
SIM = verilator
TOPLEVEL_LANG = verilog
VERILOG_SOURCES = $(shell pwd)/raytracer.sv
TOPLEVEL = fp_inv
MODULE = test_fp_inv
SIM_BUILD = sim_build_inv
EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC -GFRAC_BITS=12 -GWORD=26

include $(shell cocotb-config --makefiles)/Makefile.sim
```