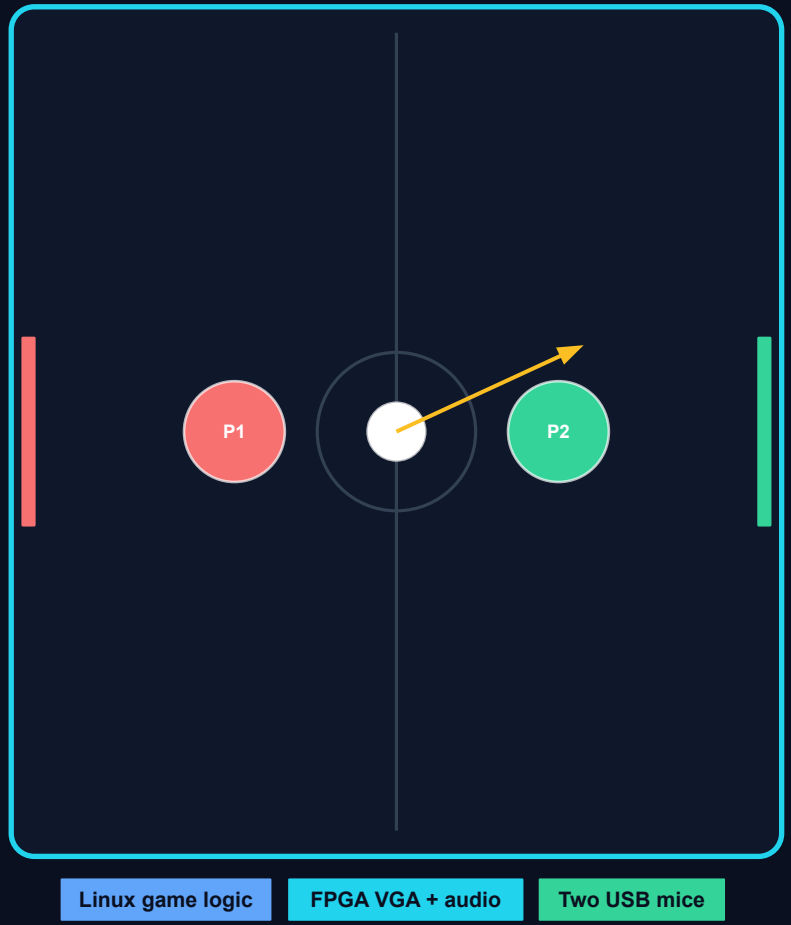
CSEE 4840: Air Hockey

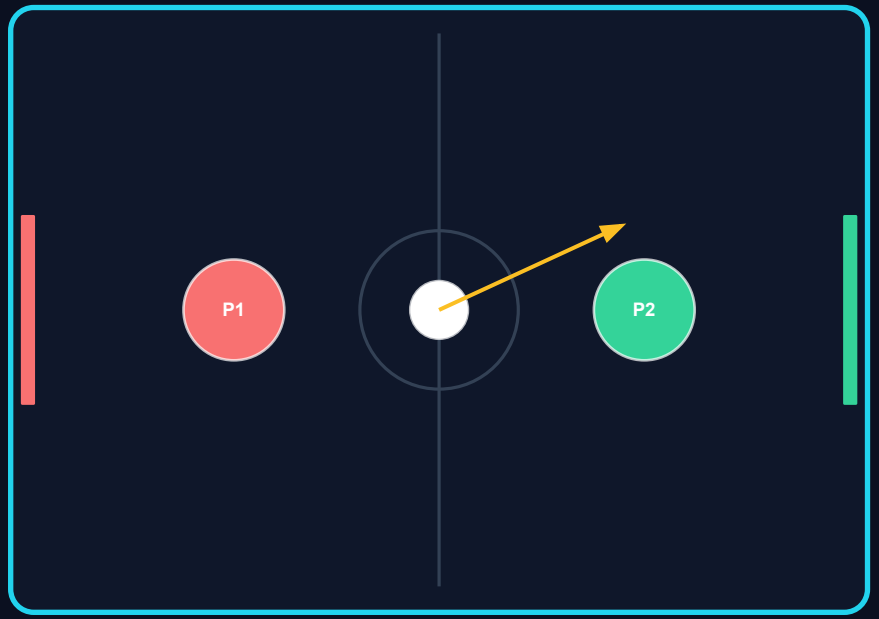
Derrick Chen · Gerald Zhao
Zening Wang · Xuepeng Han

Real-time two-player arcade game on an FPGA SoC



What the system does

The presentation focuses on engineering interfaces and the parts that are actually running.

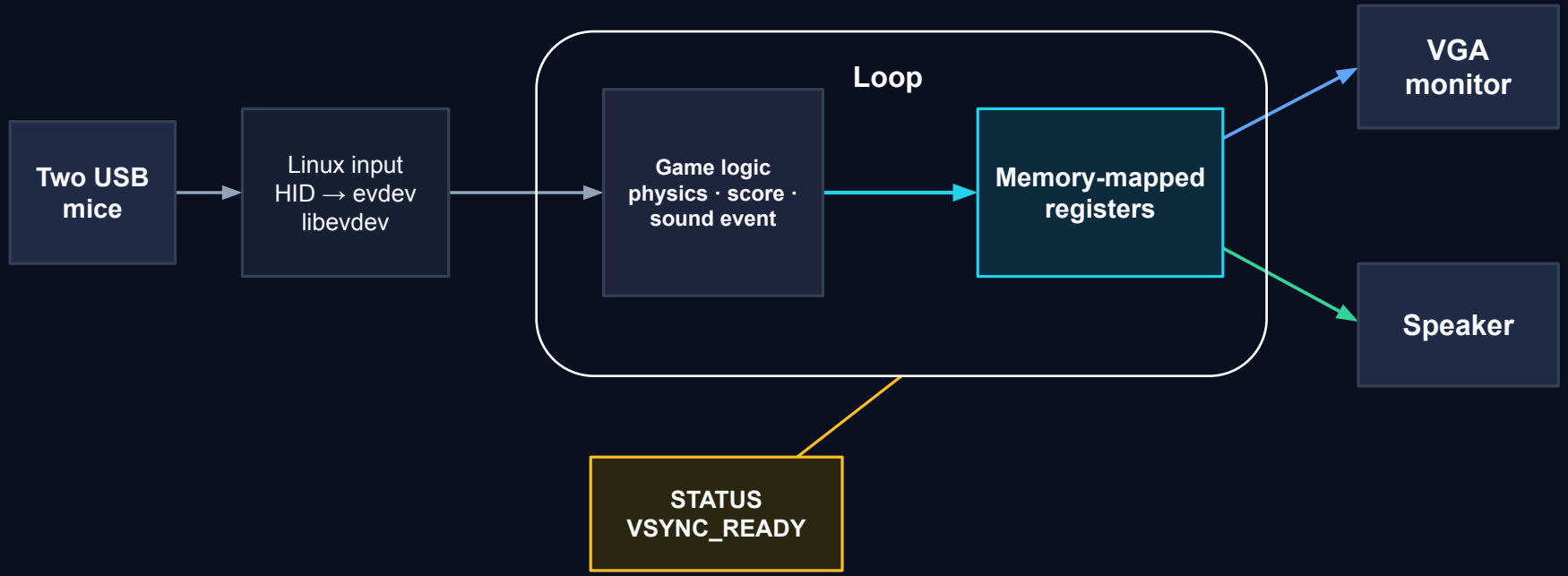


Core behaviors

- separate paddle control
- puck motion + collisions
- goals, score, sound effects
- stable display updates

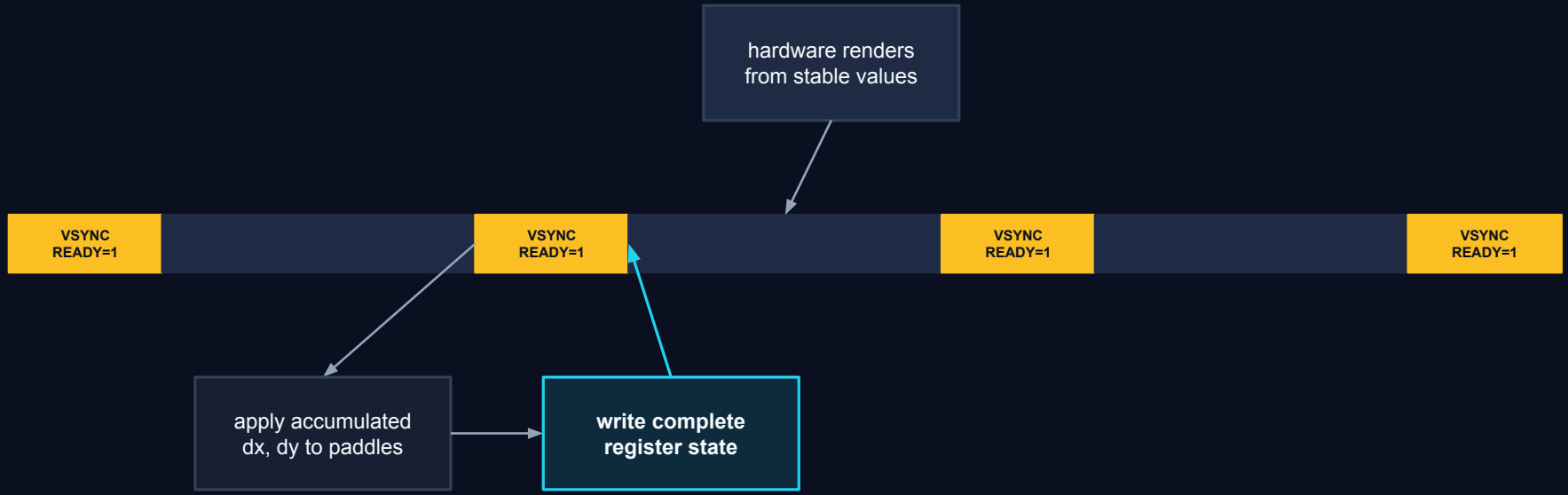
High-level block diagram

Software is the “brain”; hardware is the timed display and speaker engine.



Frame synchronization: why the image stays stable

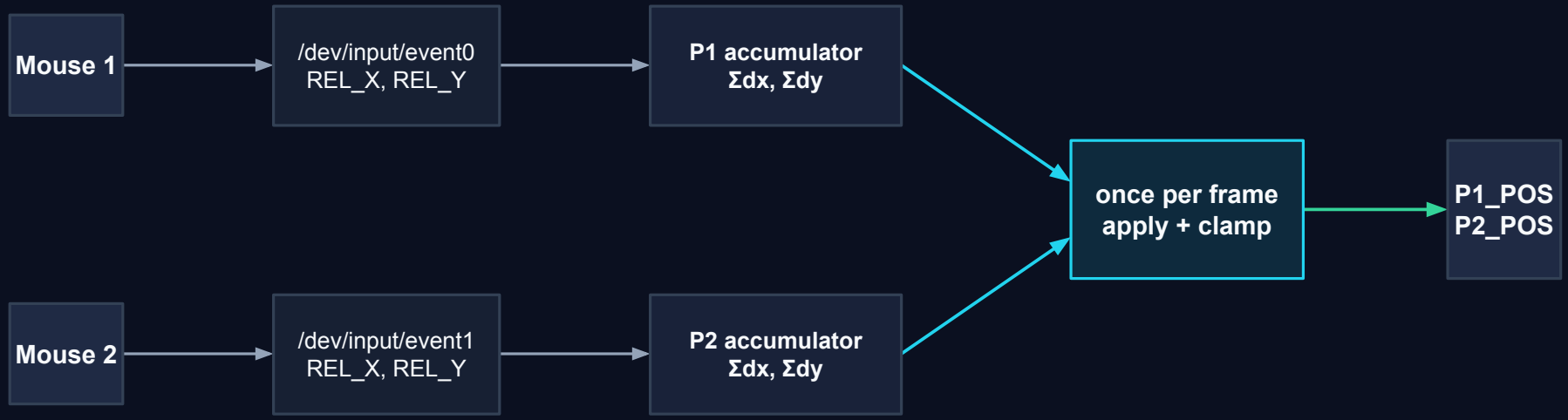
The game updates once per display frame and writes only during vertical blanking.



Key point: register writes are grouped into the safe interval, so the display does not mix old and new coordinates mid-frame.

Input path: two mice stay independent

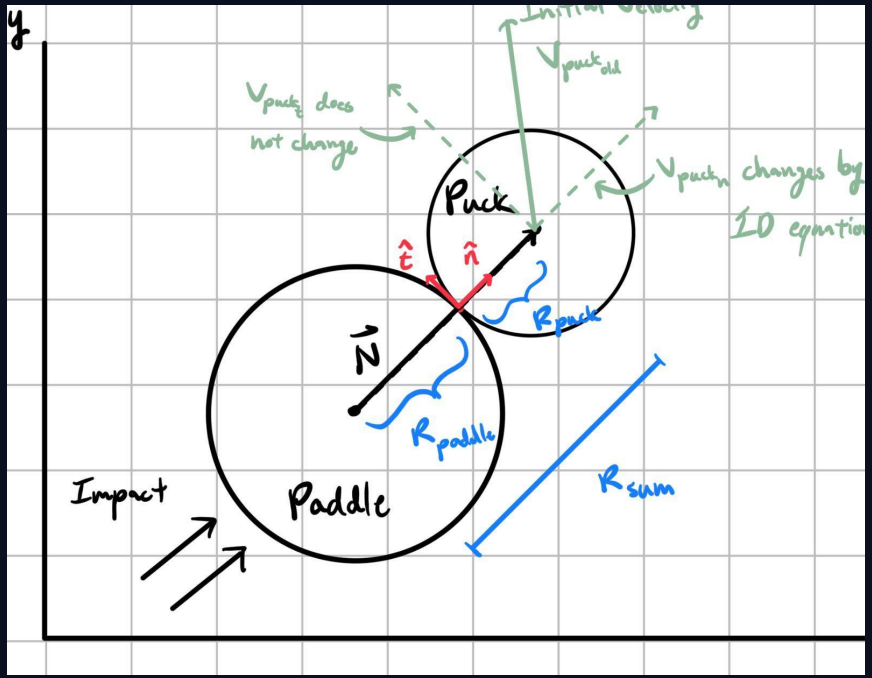
We use separate evdev devices instead of the aggregated /dev/input/mice stream.



Why this matters: each player controls only their own paddle, and mouse input rate can be faster than 60 Hz without forcing extra frame writes.

Paddle collision: solve the impact first, then update velocity

Collision time is computed inside the current frame so fast hits do not skip through objects.

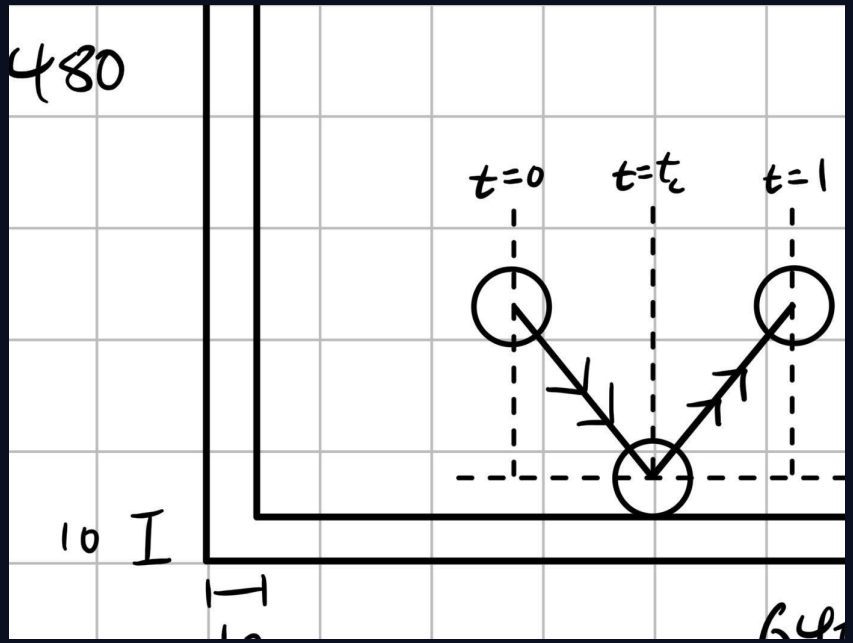


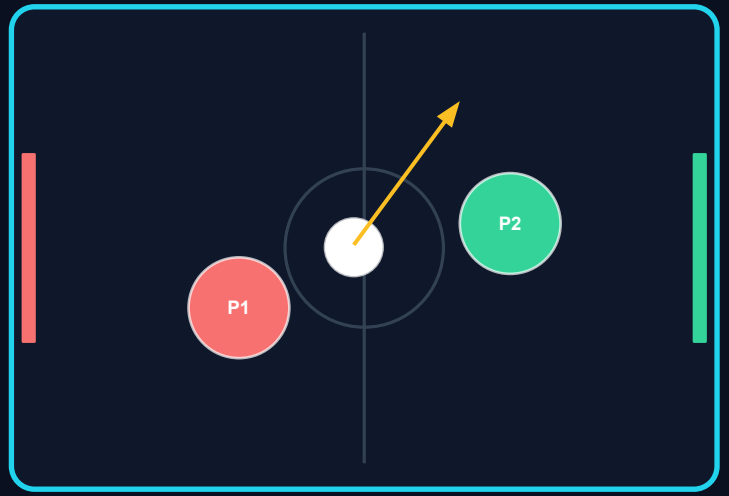
1 Calculate motion vector from the paddle's perspective → 2 Solve quadratic, tc in [0,1] choose smallest value

3 Decompose into normal and tangent. Apply 1D elastic bounce to normal → 4 Move the puck using new vectors for remaining 1 − tc time

Wall and goal logic

A side boundary can mean a bounce, a goal, or a post collision.





Flat wall → flip velocity component

Goal zone → increment score, reset state

Post edge → treat corners as stationary paddles

The wall calculation checks if tc is in [0, 1]. For goals, we check whether the puck's y-coordinate at the side boundary is inside the opening.

Register map: the hardware/software contract

Software sends complete game state; hardware turns it into pixels and sound.

Offset	Register	Software Access	Purpose
0x00	STATUS	R	VSYNC_READY status bit for safe frame update
0x04	SOUND_CONTROL	W	event code: none, wall hit, paddle hit, goal
0x08	P1_POS	W	Player 1 paddle center coordinate
0x0C	P2_POS	W	Player 2 paddle center coordinate
0x10	PUCK1_POS	W	Puck center coordinate
0x14	SCORE	W	Player 1 and Player 2 scores

Design rule: hardware does not receive draw commands. It receives state registers and renders them every frame.

Register formats: small fields, simple decoding

All interface registers are 32 bits wide. Coordinates are packed as y:x.

POSITION REGISTER

bits [31:16] y-coordinate	bits [15:0] x-coordinate

SCORE REGISTER

[31:6] reserved	[5:3] P2 score	[2:0] P1 score

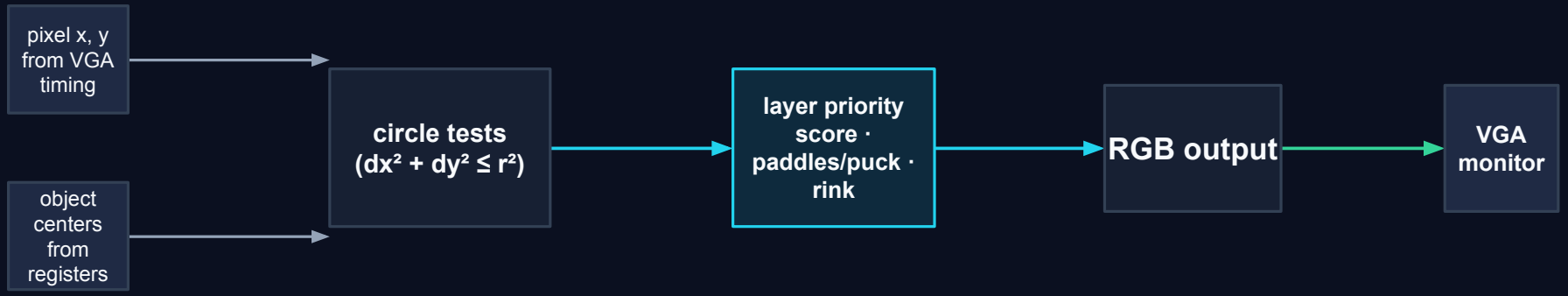
SOUND_CONTROL

[31:3] reserved	[2:0] sound event ID

STATUS bit 0 = VSYNC_READY

VGA rendering hardware

For each pixel, hardware tests simple geometry and chooses the visible layer.



Procedural graphics means puck, paddles, borders, and center line are generated by logic rather than stored as full images.

Audio hardware

Software requests a sound event; hardware streams the corresponding sample.

Event codes: 0 = none 1 = wall hit 2 = paddle hit 3 = goal

A new event can restart playback, so collision feedback stays synchronized with visible gameplay.

Resource choices

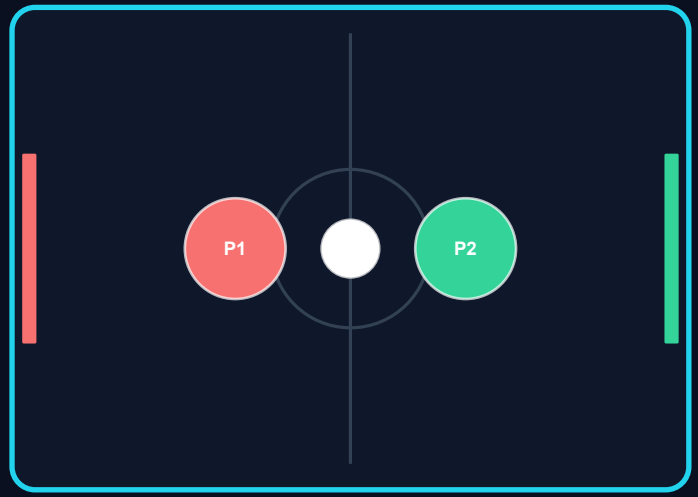
We reserve memory for audio only; circles and scores are drawn with logic.

Memory budget

3 Sound effects
205 KB

Puck / paddles / rink / score
0 graphics BRAM

Procedural shapes



circle test + comparators

This keeps the renderer simple: the CPU writes only centers and score; the FPGA computes visible pixels directly.

End-to-end frame example

One player action travels through the whole interface in one frame.

Main engineering idea: keep the interface state-based and synchronized, so software and hardware can run independently without fighting over timing.

Reference: full design block diagram

Use this only if asked to zoom into the original detailed system diagram.