Plants vs Zombies on DE1-SoC

Final design review — CSEE 4840 Embedded Systems Design

Version: V5

Columbia University · Spring 2026

A lawn-defense game, rendered by an FPGA

01

The FPGA paints every pixel.

640×480 @ 60 Hz VGA, racing-the-beam combinational compositor in SystemVerilog.
No frame buffer. No line buffer. Pixels generated as a side effect of counting clocks.

02

The HPS runs the game.

60 Hz loop in C on Linux/ARM. Plants, zombies, sun economy, win/lose checks.
A custom kernel module bridges userspace ioctls to the FPGA register file.

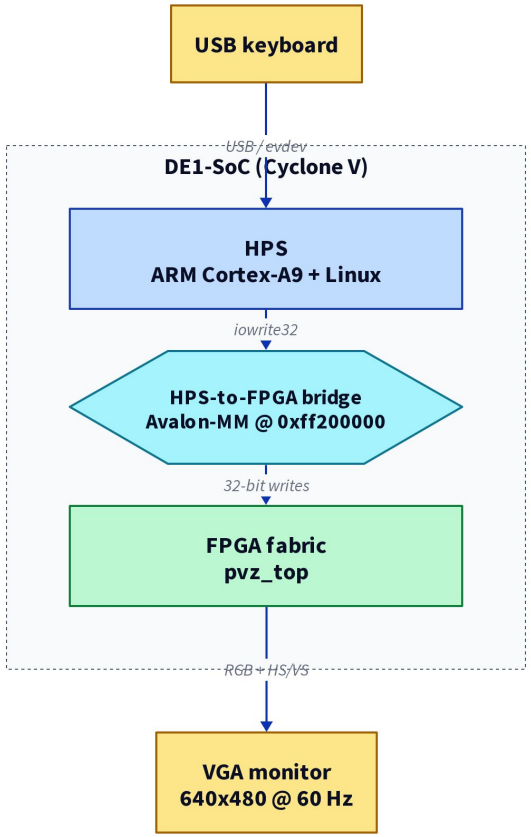
03

They share only registers.

51-word Avalon-MM register file at physical address 0xff20_0000.
No DMA. No interrupts. No vsync handshake. The two halves never synchronize.

Target: DE1-SoC, Cyclone V

Two halves on one die — ARM Cortex-A9 + FPGA fabric, joined by an on-chip bridge.



HPS

Hard Processor System

Dual-core ARM Cortex-A9 @ 925 MHz

Own DDR3 controller, USB OTG, Ethernet, SD card, UART.

Boots stock Linux 4.19 from the SD card — same way a Pi does.

FPGA

Cyclone V Fabric

LUTs, flip-flops, M10K block RAMs (10 kbit each)

SystemVerilog → Quartus Prime synth/fit → .rbf bitstream.

Programmed at boot from the SD card — or via U-Boot for fast iteration.

BUS

Lightweight HPS-to-FPGA bridge

Physical address `0xff20_0000` in the ARM's address space

From the FPGA side this bus speaks Avalon-MM — the only path we use.

All HPS→FPGA communication in this project goes through this single bridge.

Who owns what

A clean split: HPS describes the scene, FPGA draws it. Neither does the other’s job.

HPS · LINUX + C

Owns the game

Owns

- Grid state, plants, zombies, peas
- Sun economy, win/lose, frame timing
- Keyboard / gamepad input via evdev
- Decides what should be on screen

Doesn’t touch

- Pixels — no frame buffer in DDR3
- VGA signal generation
- Sprite art, palette, blanking
- Anything below 60 Hz time granularity

FPGA · SYSTEMVERILOG

Owns the pixels

Owns

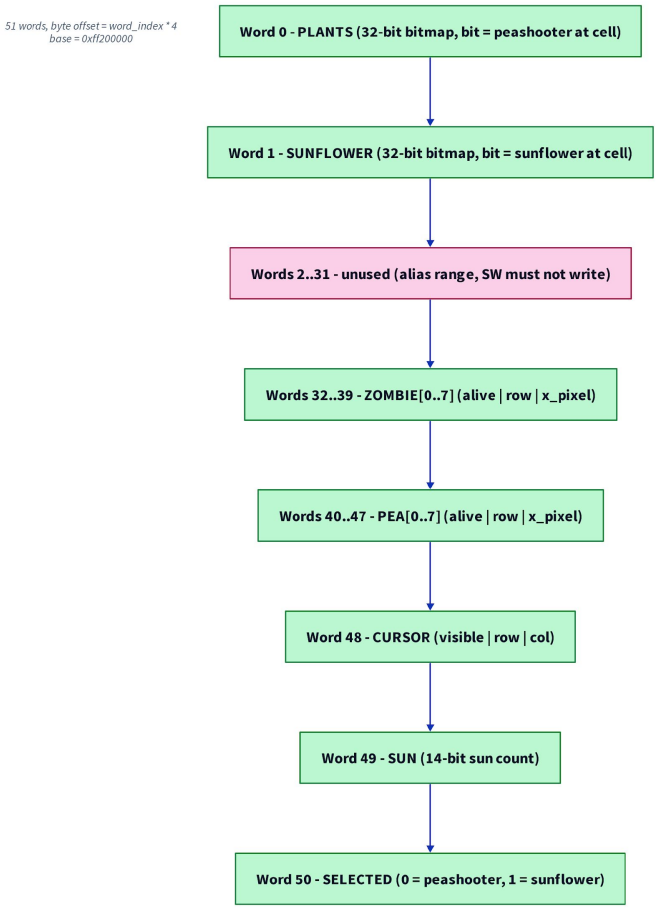
- VGA 640×480 @ 60 Hz timing
- Sprite ROMs and palette LUT
- Hit detection, layered compositing
- Reads the scene from its register file

Doesn’t touch

- Game logic, collision detection
- Any concept of time beyond “what cycle?”
- Input, randomness, sun timer
- Anything that needs to remember a frame

The entire interface contract — 51 words

No DMA, no interrupts, no vsync handshake. Software writes; hardware reads as the beam scans.



REGISTER MAP · 32-bit words

word 0 PVZ_REG_PLANTS

32-bit peashooter bitmap · bit (row×8 + col) = peashooter at that cell

word 1 PVZ_REG_SUNFLOWER

same encoding — sunflowers can't share a cell with peashooters

words 2..31 unused

words 32..39 PVZ_REG_ZOMBIE(0..7)

bit 31 = alive · [11:10] = row · [9:0] = x_pixel

words 40..47 PVZ_REG_PEA(0..7)

same encoding as zombie — 8 active projectile slots

word 48 PVZ_REG_CURSOR

bit 31 = visible · [4:2] = col · [1:0] = row

word 49 PVZ_REG_SUN

[13:0] = sun count — drives the 10-block yellow HUD

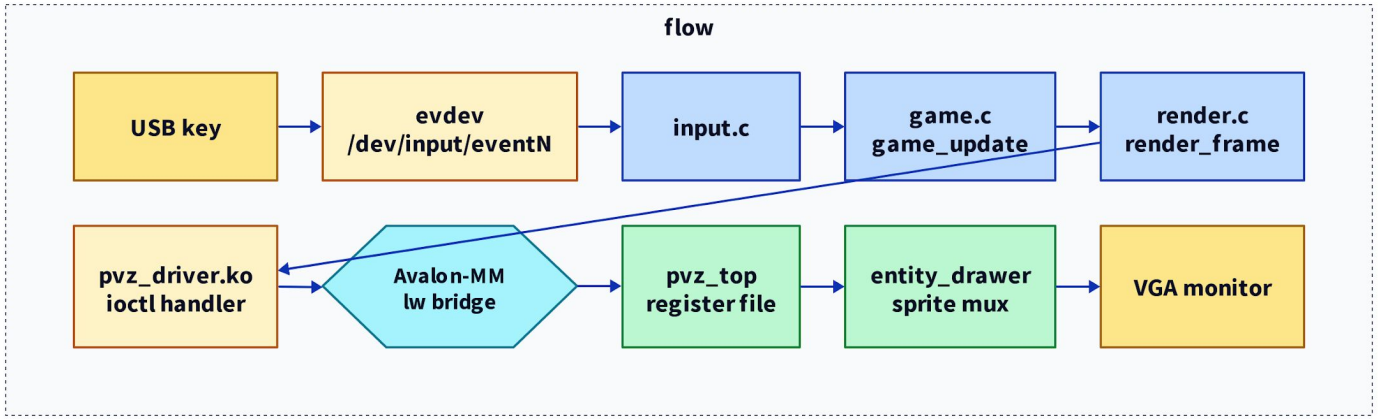
word 50 PVZ_REG_SELECTED

[1:0] = which plant the cursor wears the yellow outline on

doc/guide/diagrams/08-register-map

End-to-end frame flow

60 Hz on the HPS, 25 MHz on the FPGA. The two never synchronize.



doc/guide/diagrams/07-frame-flow

`sw/main.c:110–150` · the entire per-frame loop

```
while (gs.state >= 0) {
frame_start = get_time_usec();
process_input(&gs); // 1. poll evdev, dispatch
game_update(&gs); // 2. advance simulation one frame
render_frame(&gs); // 3. push ~20 ioctls to FPGA register file
long long elapsed = get_time_usec() - frame_start;
if (elapsed < FRAME_USEC) usleep(FRAME_USEC - elapsed); // 4. pace at 60 Hz
```

01

PART 01

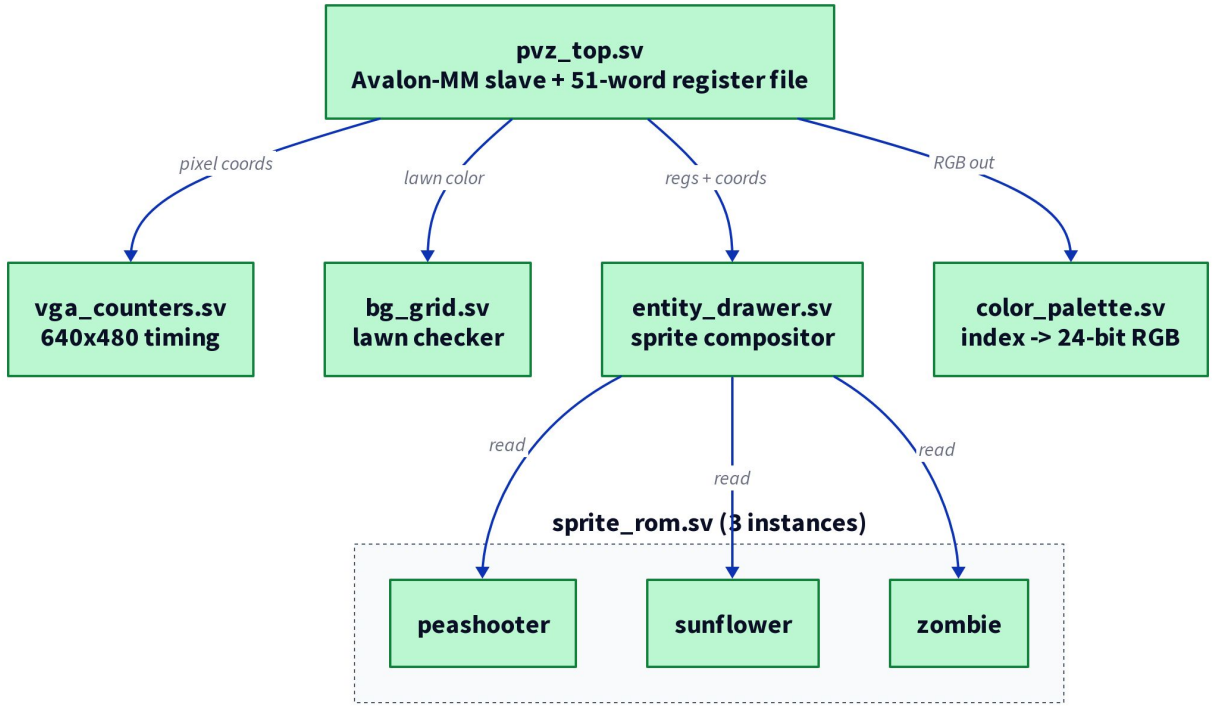
Hardware

The FPGA half — six SystemVerilog files

15 slides · approximately 15 minutes

Module hierarchy

pvz_top is the only piece of custom RTL on the fabric. It instantiates everything else.



BOARD TOP

soc_system_top.sv

DE1-SoC pin assignments • VGA conduit wiring • tie-offs.

PLATFORM DESIGNER

soc_system.qsys

Wires hps_0's lw-bridge to pvz_top_0.s1 as the only Avalon slave.

CUSTOM PERIPHERAL

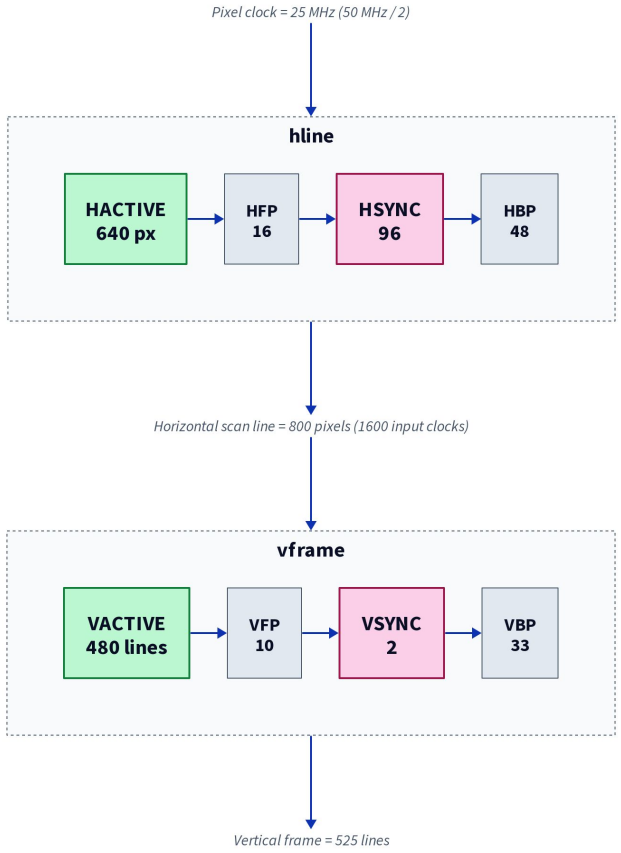
pvz_top.sv

Holds the 51-word register file. Wires the five submodules below.

```
└ vga_counters.sv
└ bg_grid.sv
└ sprite_rom.sv ×3
└ entity_drawer.sv
└ color_palette.sv
```

vga_counters — 640×480 @ 60 Hz from a 50 MHz clock

Counts half-pixels horizontally, full lines vertically. One adder, two slicers.



hw/vga_counters.sv:23–33 · timing constants

```
parameter HACTIVE = 11'd1280, // 640 visible × 2
HFRONT_PORCH = 11'd32,
HSYNC = 11'd192, // VGA_HS pulse width
HBACK_PORCH = 11'd96,
HTOTAL = HACTIVE + HFRONT_PORCH
+ HSYNC + HBACK_PORCH; // 1600

parameter VACTIVE = 10'd480,
VFRONT_PORCH = 10'd10,
VSYNC = 10'd2,
VBACK_PORCH = 10'd33,
VTOTAL = VACTIVE + VFRONT_PORCH
+ VSYNC + VBACK_PORCH; // 525
```

Pixel clock = 25 MHz, derived as hcount[0]

One VGA pixel every two 50 MHz input clocks. HTOTAL is in input clocks, not pixels.

Frame rate: 50 MHz / (HTOTAL × VTOTAL) = 59.5 Hz

Locked at compile time. Software thinks “60 Hz” — the monitor sees this.

hcount[10:1] = pixel column · vcount = pixel row

pvz_top consumes the upper bits as (px, py) for every downstream lookup.

doc/guide/diagrams/03-vga-timing

vga_counters — sync, blank, pixel clock

Two counters wrap at line / field end. Sync and blank fall out as one-line bit-slice expressions.

hw/vga_counters.sv:35–50 · counters

```
assign endOfLine = hcount == HTOTAL - 1;
assign endOfField = vcount == VTOTAL - 1;

always_ff @(posedge clk50 or posedge reset)
if (reset) hcount <= 0;
else if (endOfLine) hcount <= 0;
else hcount <= hcount + 11'd1;

always_ff @(posedge clk50 or posedge reset)
if (reset) vcount <= 0;
else if (endOfLine)
if (endOfField) vcount <= 0;
else vcount <= vcount + 10'd1;
```

hw/vga_counters.sv:52–66 · sync, blank, pixel clock

```
// Horizontal sync — active low during the pulse
assign VGA_HS = !( (hcount[10:8] == 3'b101) &
!(hcount[7:5] == 3'b111) );

// Vertical sync — one comparator
assign VGA_VS = !( vcount[9:1] == 9'd245 );

// Blank: high only in visible area
assign VGA_BLANK_n =
!( hcount[10] & (hcount[9] | hcount[8]) ) &
!( vcount[9] | (vcount[8:5] == 4'b1111) );

// 25 MHz pixel clock from 50 MHz input
assign VGA_CLK = hcount[0];
```

Sync ranges are bit-slice compares, not range comparators.

VGA_VS uses one 9-bit equality. VGA_HS combines two bit-pattern matches. The fitter turns these into trivial LUTs.

Both syncs are active-low. The monitor latches them.

VGA_BLANK_n gates the output mux in pvz_top so porches stay black instead of leaking palette colors.

bg_grid — combinational lawn checkerboard

8×4 cells of 64 px each. No clock, no state — bit slices replace all arithmetic.

THE TRICK

Cells are 64 px = 2^6

Every cell index falls out as a bit slice of the in-grid offset:

```
cell_col = gx[8:6] // 0..7
cell_row = gy[7:6] // 0..3
in_cell_x = gx[5:0] // 0..63
in_cell_y = gy[5:0] // 0..63
```

Checker parity is one XOR — no comparators, no modulos.

```
light_cell = gx[6] ^ gy[6]
```

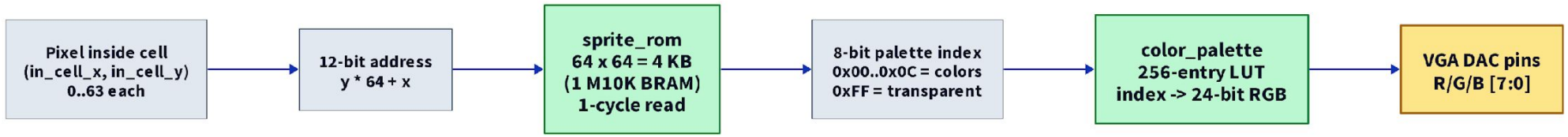
> **Maintenance hazard**
> Same constants in entity_drawer.sv:92–94 and sw/pvz.h:30–33. Hand-synced.

hw/bg_grid.sv · constants + body

```
localparam GRID_X = 10'd64;
localparam GRID_Y = 10'd112;
localparam GRID_W = 10'd512; // 8 × 64
localparam GRID_H = 10'd256; // 4 × 64

wire in_grid = (px >= GRID_X) &&
(px < GRID_X + GRID_W) &&
(py >= GRID_Y) &&
(py < GRID_Y + GRID_H);

wire [9:0] gx = px - GRID_X;
wire [9:0] gy = py - GRID_Y;

wire light_cell = gx[6] ^ gy[6];

always_comb begin
if (!in_grid) color_out = COL_BLUE;
else if (light_cell) color_out = COL_LIGHT_GREEN;
else color_out = COL_DARK_GREEN;
end
```

sprite_rom — 4096×8 synchronous block RAM

Twenty-eight lines. One M10K block. One cycle of read latency — the whole drawer is built around it.



hw/sprite_rom.sv · entire module

```
module sprite_rom #(
parameter MEM_FILE = "peashooter_idx.mem"
) (
input logic clk,
input logic [11:0] addr, // 0..4095 = y×64 + x
output logic [7:0] pixel
);

logic [7:0] rom [0:4095];

initial begin $readmemh(MEM_FILE, rom); end

always_ff @(posedge clk) pixel <= rom[addr];

endmodule
```

WHY IT MATTERS

$readmemh bakes art into bitstream

Read at synthesis time from the .mem file. To change a sprite, re-synthesize the FPGA.

always_ff → inferred M10K block

Quartus maps to one Cyclone V M10K with a registered output — 1 cycle latency.

Three instances in pvz_top.sv:179

peashooter_idx.mem, sunflower_idx.mem, zombie_idx.mem — only the parameter differs. Same module, three ROMs.

Sprite art — .mem files

64×64 palette-indexed sprites. 4096 bytes per file. One byte per pixel.

FORMAT

4096 lines. One byte per line as ASCII hex.
Line N → pixel at (x = N mod 64, y = N / 64).
Byte value: 0x00..0x0C palette index;
0xFF = transparent, the mux skips this pixel.

SAMPLE · first 8 lines of peashooter_idx.mem

```
FF    // transparent — skip in mux
FF    // transparent
07    // idx 7 = peashooter green
07    // ...
```

FILES · this branch

```
peashooter_idx.mem // 12 KB · used by plant_rom_inst
sunflower_idx.mem // 12 KB · used by sunflower_rom_inst
zombie_idx.mem // 12 KB · used by zombie_rom_inst
peas_idx.mem // 3 KB · committed but UNUSED
```

Peas are drawn procedurally in entity_drawer:
a solid 8×8 bright-green square. No ROM lookup.

PALETTE INDICES · what each byte means

00 Black	unused / default
01 Dark Green	lawn cell (dark)
02 Light Green	lawn cell (light)
03 Brown	soil / stem
04 Yellow	cursor, sun HUD, selector border
05 Red	zombie body
06 Dark Red	zombie head
07 Green	peashooter, selector-0 fill
08 Dark Green 2	peashooter stem
09 Bright Green	pea projectile (procedural)
0A White	reserved (future HUD digits)
0B Gray	reserved (future HUD bg)
0C Orange	sunflower / selector-1 fill
0D Sky Blue	outside-grid background
FF Transparent	*layer below shows through*

color_palette — 8-bit index, 24-bit RGB

A 256-entry case. 14 colors hardcoded; everything else black. No clock, no state.

hw/color_palette.sv:31–49 · case statement (excerpt)

```
always_comb begin
case (index)
8'd0: {r,g,b} = {8'h00, 8'h00, 8'h00}; // Black
8'd1: {r,g,b} = {8'h1B, 8'h5E, 8'h20}; // Dark Green
8'd2: {r,g,b} = {8'h2D, 8'h8B, 8'h2D}; // Light Green
8'd4: {r,g,b} = {8'hFF, 8'hD7, 8'h00}; // Yellow (cursor, HUD)
8'd5: {r,g,b} = {8'hFF, 8'h00, 8'h00}; // Red (zombie body)
8'd7: {r,g,b} = {8'h00, 8'h80, 8'h00}; // Green (peashooter)
8'd9: {r,g,b} = {8'h00, 8'hFF, 8'h00}; // Bright Green (pea)
8'd13: {r,g,b} = {8'h87, 8'hCE, 8'hEB}; // Sky Blue (outside)
// ... 5 more entries (indices 3, 6, 8, 10..12)
default: {r,g,b} = {8'h00, 8'h00, 8'h00};
endcase
end
```

14 used indices • 242 default to black • default protects unmapped writes.

OUTPUT MUX · pvz_top.sv:243–253

```
always_comb begin
if (VGA_BLANK_n) begin
VGA_R = pal_r;
VGA_G = pal_g;
VGA_B = pal_b;
end else begin
VGA_R = 8'h00; VGA_G = 8'h00;
VGA_B = 8'h00;
end end
```

Other forced to black

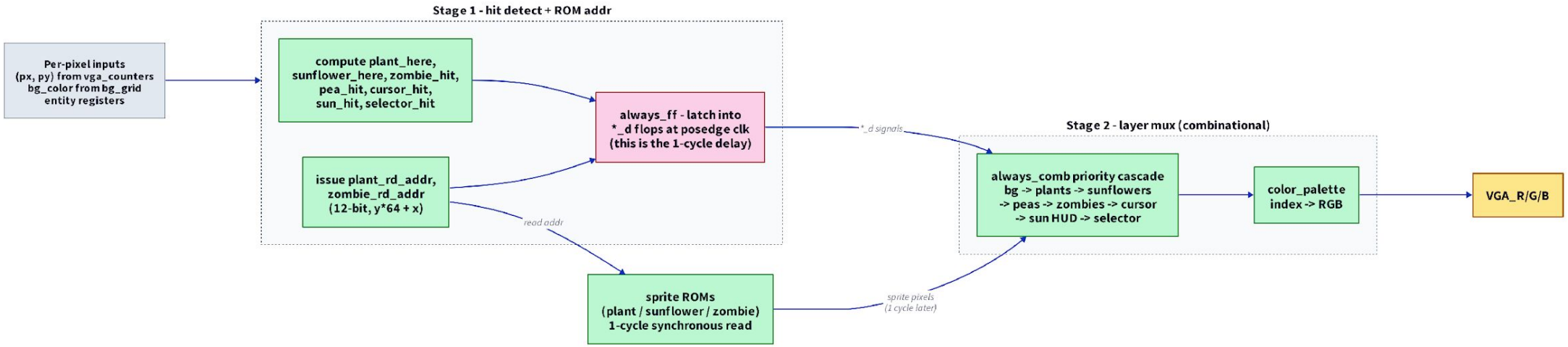
Without this, palette values would leak into HSYNC/VSYNC regions — monitor sync drift.

LUT → tiny in silicon

14 cases compile to a small ROM like multiplexer. Cost: a few dozen LUTs.

entity_drawer — the compositor

360 lines. No FSM, no frame buffer. For every (px, py), compute the color in two clocks.



doc/guide/diagrams/11-drawer-pipeline

`STAGE 1 · combinational hit detection`

For every entity type, compute “is this pixel inside this entity?” in parallel.

Issue sprite ROM addresses combinationally. ROM reads start the same cycle.

`ROM READ · 1 cycle`

M10K blocks latch address on this clock edge, pixel valid on the next.

`STAGE 2 · registered layered mux`

Hit signals registered align with the ROM output one cycle later.

Final mux paints layers bottom-up. 0xFF = transparent, layer underneath wins.

`END-TO-END LATENCY`

2 cycles between (px, py) and color_out — invisible at the 25 MHz pixel rate.

Stage 1 — grid math, ROM address

(px, py) → cell index → in-cell offset → sprite ROM address. All in one combinational cone.

hw/entity_drawer.sv:140–166

```
// inside the lawn?
wire in_grid_x = (px >= GRID_X) &&
(px < GRID_X + 10'd512);
wire in_grid_y = (py >= GRID_Y) &&
(py < GRID_Y + 10'd256);
wire in_grid = in_grid_x && in_grid_y;

// in-grid offset
wire [9:0] gx = px - GRID_X;
wire [9:0] gy = py - GRID_Y;
// bit-slice cell indices (cells are 64 =
2^6)
wire [2:0] cell_col = gx[8:6];
wire [1:0] cell_row = gy[7:6];
wire [5:0] in_cell_x = gx[5:0];
wire [5:0] in_cell_y = gy[5:0];
// plant / sunflower bit for this
cell
wire [4:0] plant_idx = {cell_row, cell_col};
wire plant_here = in_grid && plant_present[plant_idx];
wire sunflower_here = in_grid && sunflower_present[plant_idx];

// sprite ROM address — 1:1 from cell-local pixel
assign plant_rd_addr = {in_cell_y, in_cell_x};
```

NOTES

Every divide is a shift

cell_col = gx / 64 ≡ gx[8:6]
in_cell_x = gx mod 64 ≡ gx[5:0]

Sunflower ROM shares the address

Both plant ROMs receive plant_rd_addr.
Only one of plant_here / sunflower_here is true per cell — stage 2 picks the survivor.

{cell_row, cell_col} packs a cell index

5 bits = row(2) || col(3). Index into the 32-bit bitmap from PVZ_REG_PLANTS.

{in_cell_y, in_cell_x} packs a ROM addr

12 bits = y(6) || x(6). Sprite is 64×64 — offset within the cell IS the ROM address.

No multiplier, no divider, no FSM

The whole cone is a few LUTs and one subtract.

Stage 1 — zombie priority encoder

Eight slots. First alive zombie covering this pixel wins. Pixel offset becomes the ROM address.

hw/entity_drawer.sv:173–194

```
logic zombie_hit_comb;
logic [5:0] zombie_in_x, zombie_in_y;

always_comb begin
zombie_hit_comb = 1'b0;
zombie_in_x = 6'd0;
zombie_in_y = 6'd0;
for (int i = 0; i < 8; i++)
begin
logic [9:0] zy_top, dx, dy;
zy_top = GRID_Y + ({8'd0, zombie_row[i]} << 6);
dx = px - zombie_x[i];
dy = py - zy_top;
if (zombie_alive[i] && !zombie_hit_comb &&
px >= zombie_x[i] && px < zombie_x[i] + ZOMBIE_W &&
py >= zy_top && py < zy_top + ZOMBIE_H)
begin
zombie_hit_comb = 1'b1;
zombie_in_x = dx[5:0];
zombie_in_y = dy[5:0];
end
end
end

assign zombie_rd_addr = {zombie_in_y, zombie_in_x};
```

NOTES

!zombie_hit_comb is the priority gate

Slot 0 has highest priority — the loop unrolls to 8 cascaded conditionals.

Why priority doesn't matter in practice

Zombies stay in their own rows and don’t overlap. If two ever shared a pixel, slot 0 would win — not a visible issue at 5 zombies.

64×64 bounding box, 6-bit pixel offset

dx[5:0] / dy[5:0] = position inside the sprite. Combined into the 12-bit ROM address — same pattern as plant_rd_addr in P16.

Synthesis: 8 parallel comparators

for-loop unrolls at compile time. Each slot’s bounding-box check runs in parallel; priority cascade is a chain of AND gates.

Stage 1 — peas, cursor, sun HUD

Three procedural geometries. No ROM. No sprite art. All combinational.

PEAS · entity_drawer.sv:199–211

```
logic pea_hit_comb;
always_comb begin
pea_hit_comb = 1'b0;
for (int i=0; i<8; i++)
begin
 logic [9:0] py_top;
py_top = GRID_Y +
({8'd0, pea_row[i]} << 6)
+ 10'd28;
if (pea_alive[i] &&
px >= pea_x[i] &&
px < pea_x[i]+PEA_SIZE &&
py >= py_top &&
py < py_top+PEA_SIZE)
pea_hit_comb = 1'b1;
end
end
```

8×8 solid bright-green

OR-reduced — one bool suffices.
+28 px y-offset = vertical center of row.
Color is hardcoded, no ROM read.

CURSOR · entity_drawer.sv:213–230

```
logic cursor_hit_comb;
always_comb begin
logic [9:0] cur_left, cur_top;
cur_left = GRID_X +
({7'd0, cursor_col} << 6);
cur_top = GRID_Y +
({8'd0, cursor_row} << 6);
cursor_hit_comb = 1'b0;
if (cursor_visible &&
px >= cur_left &&
px < cur_left+CELL && ...) begin
if ((px - cur_left) <
CURSOR_BORDER || ...)
cursor_hit_comb = 1'b1;
end
end
```

Hollow yellow rectangle

Inside cell, within 4 px of any edge.
Doesn’t obscure plants beneath.
Hidden by software on WIN/LOSE.

SUN HUD · entity_drawer.sv:266–280

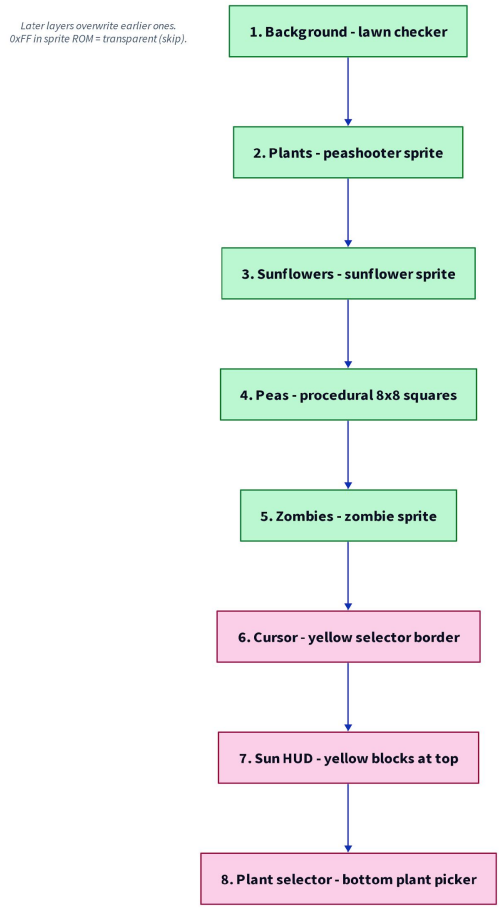
```
logic sun_hit_comb;
always_comb begin
sun_hit_comb = 1'b0;
if (py >= SUN_Y &&
py < SUN_Y+SUN_BH) begin

for (int i=0; i<10; i++)
begin
logic [9:0] bx;
bx = SUN_X +
10'(i) * SUN_PITCH;
if (sun_value >=
14'((i+1) * 50)
&&
px >= bx &&
px < bx + SUN_BW)
sun_hit_comb = 1'b1;
end end
end
```

10 yellow blocks — 50 sun each

Block i lit when sun ≥ (i+1) × 50.
500 sun maxes the display;
register still counts higher internally.

Stage 2 — layered mux

Paint bottom-up. 0xFF transparency lets the layer below show through.



hw/entity_drawer.sv:330–358 · final mux (condensed)

```
always_comb begin
color_out = bg_color_d; // 1. lawn

if (plant_here_d && plant_rd_pixel != COL_TRANSPARENT)
color_out = plant_rd_pixel; // 2. peashooter
if (sunflower_here_d && sunflower_rd_pixel != COL_TRANSPARENT)
color_out = sunflower_rd_pixel; // 3. sunflower
if (pea_hit_d) color_out = COL_BRIGHT_GREEN; // 4. pea

if (zombie_hit_d && zombie_rd_pixel != COL_TRANSPARENT)
color_out = zombie_rd_pixel; // 5. zombie
if (cursor_hit_d) color_out = COL_YELLOW; // 6. cursor
if (sun_hit_d) color_out = COL_YELLOW; // 7. sun HUD
if (sel0_hit_d) color_out = COL_GREEN; // 8. selector 0 fill
if (sel1_hit_d) color_out = COL_ORANGE; // 9. selector 1 fill

if (sel0_border_d && selected_plant_d == 2'd0)
color_out = COL_YELLOW; // 10. border 0
if (sel1_border_d && selected_plant_d == 2'd1)
color_out = COL_YELLOW; // 10. border 1
end
```

> **Reads bottom-up, last write wins**
>
> Order in source = Z-order on screen. The transparency check protects layers from each other. Selector border is gated by selected_plant_d — TAB toggles which icon wears it.

doc/guide/diagrams/05-entity-layers

pvz_top — Avalon-MM write decoder

One always_ff block latches all 51 registers. Decoder unrolls per the chapter-06 map.

hw/pvz_top.sv:95–148 · address decode (reset block omitted)

```
always_ff @(posedge clk or posedge reset) begin
if (reset) begin /* zero every register */ end
else if (chipselect && write) begin

if (address == 6'd0) plant_present <= writedata;
else if (address == 6'd1) sunflower_present <= writedata;

else if (address < 6'd40) begin // zombies, words 32..39
zombie_alive[address[2:0]] <= writedata[31];
zombie_x[address[2:0]] <= writedata[9:0];
zombie_row[address[2:0]] <= writedata[11:10];
end

else if (address < 6'd48) begin // peas, words 40..47
pea_alive[address[2:0]] <= writedata[31];
pea_x[address[2:0]] <= writedata[9:0];
pea_row[address[2:0]] <= writedata[11:10];
end

else if (address == 6'd48) begin
cursor_visible <= writedata[31];
cursor_col <= writedata[4:2];
cursor_row <= writedata[1:0];
end

else if (address == 6'd49) sun_value <= writedata[13:0];
else if (address == 6'd50) selected_plant <= writedata[1:0];
```

GOTCHA

address is in WORDS

addressUnits = WORDS in pvz_top_hw.tcl. CPU writes byte offset; Avalon delivers it as a word index. Kernel must multiply by 4 (see pvz_driver.c:40).

address[2:0] indexes 8-element arrays

Low 3 bits of word index select which zombie / pea slot to write. Cheap decode — the high bits selected the slot type already.

No write strobes, no readback

Slave has no read path. Software re-writes every register every frame — no delta tracking.

pvz_top — submodule wiring

VGA timing → bg_grid → three sprite ROMs → entity_drawer → color_palette. One pipeline.

hw/pvz_top.sv:53–241 · condensed instantiation chain

```
// VGA timing — produces hcount, vcount, sync, blank
vga_counters counters(.clk50(clk), .reset(reset), .hcount, .vcount, ... );
wire [9:0] px = hcount[10:1]; // upper bits = pixel x (0..639)
wire [9:0] py = vcount; // y is 1 tick/line
already
// Combinational background
bg_grid bg_inst(.px(px), .py(py), .color_out(bg_color));

// Three sprite ROMs — same module, parameterized by .mem file
sprite_rom #(.MEM_FILE("peashooter_idx.mem")) plant_rom_inst(
.clk(clk), .addr(plant_addr), .pixel(plant_pixel));
sprite_rom #(.MEM_FILE("sunflower_idx.mem")) sunflower_rom_inst(
.clk(clk), .addr(plant_addr), .pixel(sunflower_pixel));
sprite_rom #(.MEM_FILE("zombie_idx.mem")) zombie_rom_inst(
.clk(clk), .addr(zombie_addr), .pixel(zombie_pixel));

// Pack 8-element unpacked arrays into wide buses for the drawer
generate for (gi = 0; gi < 8; gi++) begin : pack_entities
assign zombie_x_packed[gi*10 +: 10] = zombie_x[gi];
assign pea_x_packed [gi*10 +: 10] = pea_x [gi];
...
end endgenerate

// Compositor → palette → output mux (P14 covers the blanking gate)
entity_drawer drawer_inst( /* ~20 ports */ );
color_palette pal_inst(.index(pixel_color), .r(pal_r), .g(pal_g), .b(pal_b));
```

NOTES

Plant ROMs share the address bus

Peashooter and sunflower ROMs both take plant_addr — the cell-local offset. Stage 2 picks the right one.

Why packed buses?

Some synthesis tools reject unpacked arrays across module ports. Generate-loop packs them on the fly — zero runtime cost (pure wiring).

No vsync handshake

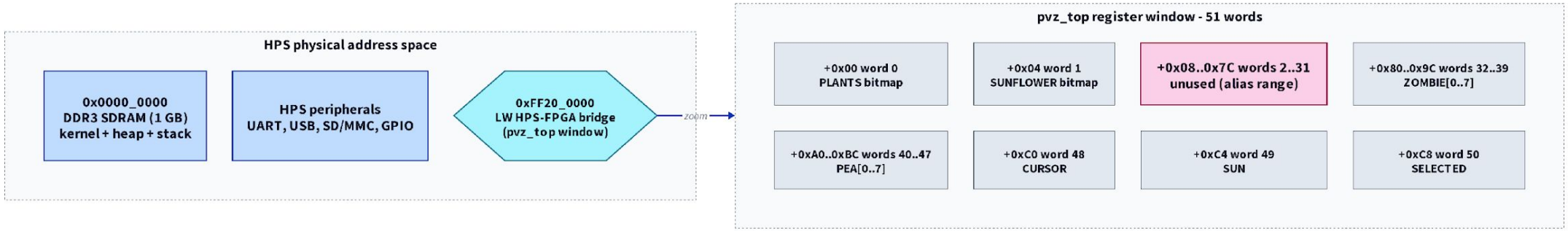
Entity register file is plain D flops — no shadow/active pair. Writes take effect on the next clock edge.

End-to-end latency: 2 cycles

vga_counter → bg_grid (0) → ROM read (1) → mux (2) → palette → VGA DAC.

Memory map — bytes meet words

CPU thinks bytes. Avalon delivers words. The kernel multiplies by 4 in one place.



doc/guide/diagrams/10-memory-map

HARDWARE SIDE · pvz_top_hw.tcl:23

```
set_module_assignment
embeddedsw.dts.compatible "csee4840,pvz_gpu-1.0"
```

SOFTWARE SIDE · pvz_driver.c:111–115

```
static const struct of_device_id pvz_of_match[] = {
{ .compatible = "csee4840,pvz_gpu-1.0" },
{}, };
```

These two strings must match exactly

qsys-generate writes the .tcl string into the device tree; the kernel looks up matching drivers by string.
Mismatch → pvz_probe never fires → /dev/pvz never appears → userspace open() returns ENOENT.

02

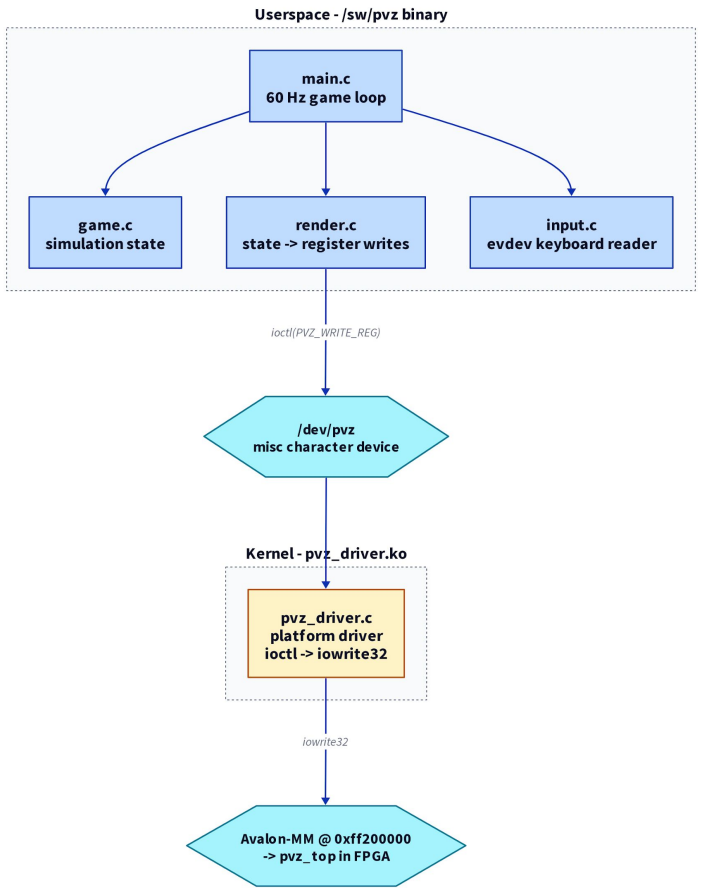
PART 02

Software

The HPS half — kernel module + userspace game

Software modules + shared ABI

One header (pvz.h) shared between kernel module and userspace. Single source of truth.



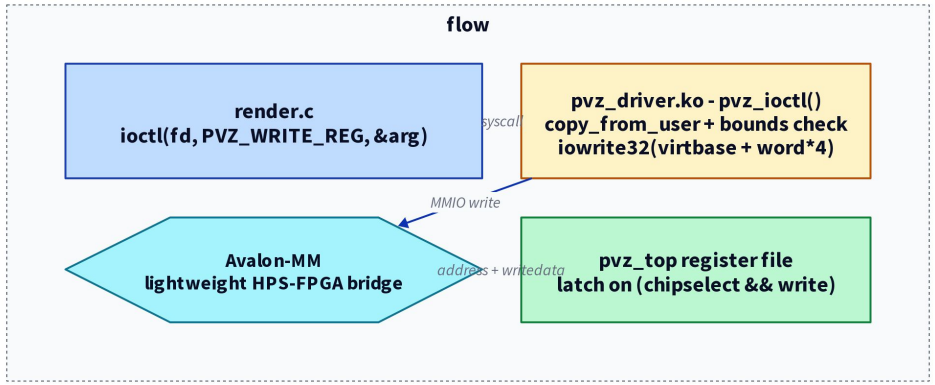
sw/pvz.h · the contract (excerpt)

```c
/* Word indices in the register file
*/
#define PVZ_REG_PLANTS 0
#define PVZ_REG_SUNFLOWER 1
#define PVZ_REG_ZOMBIE(idx) (32 + (idx))
#define PVZ_REG_PEA(idx) (40 + (idx))
#define PVZ_REG_CURSOR 48
#define PVZ_REG_SUN 49
#define PVZ_REG_SELECTED 50
#define PVZ_NUM_REGS 51

/* Pack a zombie/pea word */
static inline unsigned int
pvz_pack_entity(int alive, int row, int x_pixel) {
return ((unsigned)(alive & 1) << 31) |
((unsigned)(row & 3) << 10) |
((unsigned)(x_pixel & 0x3FF));
}

/* ioctl ABI — one struct, one command */
typedef struct {
unsigned int word_index; /* 0..50 */
unsigned int value;
} pvz_write_arg_t;

#define PVZ_WRITE_REG _IOW('p', 1, pvz_write_arg_t)

// Both halves #include this file. Bit layouts match HW exactly.
```

doc/guide/diagrams/06-sw-modules

pvz_driver.c — probe + ioctl

145 lines. Platform driver + miscdevice + one ioctl. /dev/pvz appears on probe success.



doc/guide/diagrams/14-ioctl-path

PROBE PATH

DT node compatible matches → `pvz_probe()`
misc_register → of_address_to_resource →
request_mem_region → of_iomap
→ virtbase now usable by iowrite32.

VERIFY ON BOARD

```
dmesg | tail • cat /proc/iomem
```

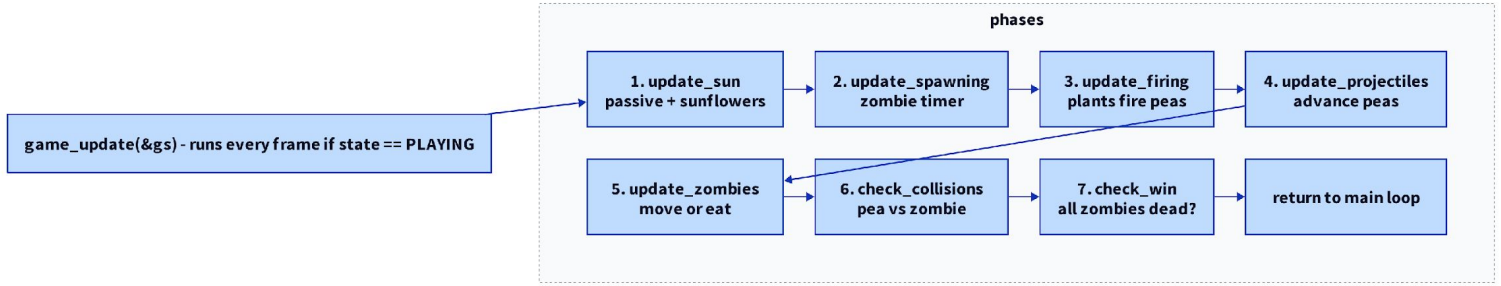
`sw/pvz_driver.c:30–48 · the entire write path`

```
static long pvz_ioctl(struct file *f,
unsigned int cmd,
unsigned long arg)
{
pvz_write_arg_t w;

switch (cmd) {
case PVZ_WRITE_REG:
if (copy_from_user(&w,
(pvz_write_arg_t *)arg, sizeof(w)))
return -EACCES;

if (w.word_index >= PVZ_NUM_REGS)
return -EINVAL;

iowrite32(w.value,
dev.virtbase + (w.word_index * 4));

break;

default:
return -EINVAL;
}
return 0;
}
```

> **word_index × 4 = byte offset**
> iowrite32 takes bytes; Avalon takes words. Bridged here.

game.c — per-frame pipeline + FSM

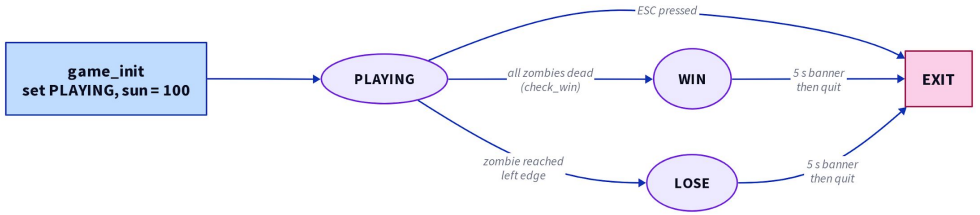
Seven helpers in a fixed order. PLAYING is the only state that runs them.



sw/game.c:288–302 · game_update body

```
void game_update(game_state_t *gs)
{
if (gs->state != STATE_PLAYING) return;
gs->frame_count++;

update_sun(gs); // 1
update_spawning(gs); // 2 spawn before move
update_firing(gs); // 3
update_projectiles(gs); // 4
update_zombies(gs); // 5
check_collisions(gs); // 6 after both moves
check_win(gs); // 7
}
```

STATE MACHINE



Terminal → main loop prints banner, sleeps 5 s, exits.
state = -1 sentinel (ESC) breaks the loop before render_frame.

render.c — game state → register file

Walk the struct, build words, fire ioctls. About 20 per frame. No diffing, no compaction.

sw/render.c:32–47 · render_plants · pack 32 cells → 2 bitmaps

```
static void render_plants(const game_state_t *gs) {
uint32_t pea_bits = 0;
uint32_t sun_bits = 0;
for (int r = 0; r < GRID_ROWS; r++) {
for (int c = 0; c < GRID_COLS; c++) {
int t = gs->grid[r][c].type;
if (t == PLANT_PEASHOOTER)
pea_bits |= (1u << (r * 8 + c));
else if (t == PLANT_SUNFLOWER)
sun_bits |= (1u << (r * 8 + c));
} }
write_reg(PVZ_REG_PLANTS, pea_bits);
write_reg(PVZ_REG_SUNFLOWER, sun_bits);
}
```

All 32 plant cells pack into two 32-bit bitmaps.

Two ioctls cover the entire lawn — the densest part of the protocol.

No delta tracking. Every register, every frame.

~20 ioctls · dozens of microseconds · cheaper than tracking dirty bits.

sw/render.c:55–66 · render_zombies · 8-slot mirror

```
static void render_zombies(const game_state_t *gs) {
for (int i = 0; i < PVZ_MAX_ZOMBIES; i++) {
int alive = 0, row = 0, x = 0;
if (i < MAX_ZOMBIES &&
gs->zombies[i].active) {
alive = 1;
row = gs->zombies[i].row;
x = gs->zombies[i].x_pixel;
}
write_reg(PVZ_REG_ZOMBIE(i),
pvz_pack_entity(alive, row, x));
}
}
```

Hardware slot i mirrors game slot i.

No compaction. Slot inactive → write 0 → alive bit clear hides sprite.

Peas use a different policy — truncating.

Sim has 16 slots, HW has 8 — render_peas walks active & copies first 8.

input.c — evdev to action codes

One source for keyboard AND Xbox 360-style gamepad. Same INPUT_* codes consumed by main.c.

sw/input.c:31–71 · input_poll (condensed)

```
int input_poll(void) {
struct input_event ev;
while (read(input_fd, &ev, sizeof(ev)) == sizeof(ev)) {

// xpad reports D-pad as ABS hat axis on some variants
if (ev.type == EV_ABS) {
if (ev.code == ABS_HAT0X) {
if (ev.value < 0) return INPUT_LEFT;
if (ev.value > 0) return INPUT_RIGHT;
} else if (ev.code == ABS_HAT0Y) {
if (ev.value < 0) return INPUT_UP;
if (ev.value > 0) return INPUT_DOWN;
}
continue;
}

if (ev.type != EV_KEY || ev.value != 1)
continue;
switch (ev.code) {
case KEY_UP: case BTN_DPAD_UP: return INPUT_UP;
case KEY_DOWN: case BTN_DPAD_DOWN: return INPUT_DOWN;
case KEY_LEFT: case BTN_DPAD_LEFT: return INPUT_LEFT;
case KEY_RIGHT: case BTN_DPAD_RIGHT: return INPUT_RIGHT;
case KEY_SPACE: case BTN_SOUTH: return INPUT_SPACE;
case KEY_D: case BTN_EAST: return INPUT_D;
case KEY_ESC: case BTN_START: return INPUT_ESC;
case KEY_TAB: case BTN_TL: case BTN_TR: return INPUT_TAB;
} }
```

MAPPING · one action, two devices

ACTION	KEYBOARD	XBOX 360
INPUT_UP	Up arrow	DPAD_UP
INPUT_DOWN	Down arrow	DPAD_DOWN
INPUT_LEFT	Left arrow	DPAD_LEFT
INPUT_RIGHT	Right arrow	DPAD_RIGHT
INPUT_SPACE	Space	A (BTN_SOUTH)
INPUT_D	D	B (BTN_EAST)
INPUT_ESC	Esc	Start
INPUT_TAB	Tab	LB / RB

Non-blocking, polled once per frame

O_NONBLOCK fd; read returns EAGAIN when empty. Main loop drains queued events.

Press only (value == 1)

Releases are dropped. No key-repeat handling — cursor moves once per press, on purpose.

Resource & Timing

Resource / Metric	Used	Total	%	Notes
ALMs (logic)	605	32,070	2 %	Plenty of headroom
Registers (flip-flops)	784	—	—	Game state + pipeline regs
Memory bits	98,304	4,065,280	2 %	**= 3 × 64×64 × 8 bit ✓**
M10K blocks (BRAM)	12	397	3 %	4 blocks per sprite ROM
DSP blocks	0	87	0 %	All geometry via bit-slicing
PLLs	0	6	0 %	25 MHz from `hcount[0]`
Fmax (clock_50_1)	**92.0 MHz**	—	—	Slow 1100 mV 85 °C (sign-off)
Fmax (clock_50_1)	92.52 MHz	—	—	Slow 1100 mV 0 °C

Thank You!