Maze Chase Game

A maze-pursuit game on the DE1-SoC

CSEE 4840 – Embedded System Design – Spring 2026

Final Project Report

Ryan Lo	yl5972 – VGA pipeline, compositor, register file
Junhao Qu	jq2434 – Maze ROM, tile timers, gen_mif
Boxiong Li	bl3155 – Qsys integration, build flow, demo
Kevin Liu	kl3755 – Kernel driver, BFS AI, USB keyboard

Columbia University
Prof. Stephen A. Edwards

May 13, 2026

Project source code is included in Appendix A.

Contents

1 Project Overview

1.1 What we built

Maze Chase is a real-time maze pursuit game running on the Terasic DE1-SoC. A custom Avalon-MM peripheral on the Cyclone V FPGA draws a 30 × 20 tile maze at 640 × 480 / 60 Hz over VGA, and a small Linux application on the HPS supplies all the game logic.

You play the Evader, a single token in the middle of the maze. Two ghosts spawn in the upper corners and chase you using breadth-first-search pathfinding. Six *Special Tiles* are scattered around the maze; standing on one for ≈ 3 seconds activates it, and activating any three of the six unlocks the Gate Tile at the top of the maze. Reach the gate to win. Get caught by a ghost to lose.

1.2 Why this is the right shape for the class

The course rubric pushes for a balanced hardware–software split with at least one non-trivial custom peripheral. Maze Chase fits that shape naturally:

- The **hardware** job (drawing a 60 Hz VGA frame, sampling the player position at exactly the right pixel cycles, counting frames-on-tile without scheduling jitter) is the kind of work an FPGA is uniquely good at.
- The **software** job (BFS pathfinding, USB input, win/loss state, restart) is naturally a procedural job that the Cortex-A9 + Linux combination handles in microseconds with code we can read at a glance.

The Avalon-MM register file is the contract between them. Twelve 32-bit registers (in a 16-word address space) carry every piece of state the two sides share: positions, status bits, per-tile frame counts, and the frame counter the HPS uses to pace its loop.

1.3 What changed from the proposal

Three changes are worth flagging up front because they shape the rest of the report:

1. **No separate pixel-clock PLL.** The original proposal called for a 25.175 MHz pixel clock from a dedicated PLL. The lab3 vga_counters reference shows you can clock the whole pipeline at 50 MHz and use hcount[0] as the 25 MHz strobe to the DAC. We adopted the same idea, which removed an entire clock domain and a CDC boundary.
2. **Active-high synchronous reset, 4-bit Avalon address.** The proposal had `reset_n` and a 6-bit word address. Both changed during Qsys bring-up: the lab3 starter and our Qsys reset network were friendlier with active-high reset, and we settled on twelve registers (so a 4-bit address bus covering 16 words is enough). The implementation is consistent throughout.
3. **libusb, not `/dev/input/eventX`.** The proposal sketched a usbhid + evdev path. The Logitech keyboard we ended up with is a composite HID device whose arrow keys appear on the Boot subdevice but whose Linux input-layer mapping was inconsistent across reboots. We switched to a tiny libusb HID-Boot reader (lifted from lab2) running on a dedicated pthread, which works without surprises.

1.4 Demo footprint

After full Quartus compile (Fitter, Spring 2026 build of Quartus Prime Lite 20.1):

Resource	**Used / Available**
ALMs	397 / 32,070 (1.2%)
M10K blocks	2 / 397 (0.5%)
Total block memory	$\sim$ 4.1 kbit
DSP blocks	0 / 87
PLLs	0 / 6
Worst-case Fmax (50 MHz net)	141.62 MHz (1100 mV / 85 °C corner)

Comfortably small. Timing meets with a 92 MHz margin against the 50 MHz requirement.

2 System Architecture

2.1 Top-level block diagram

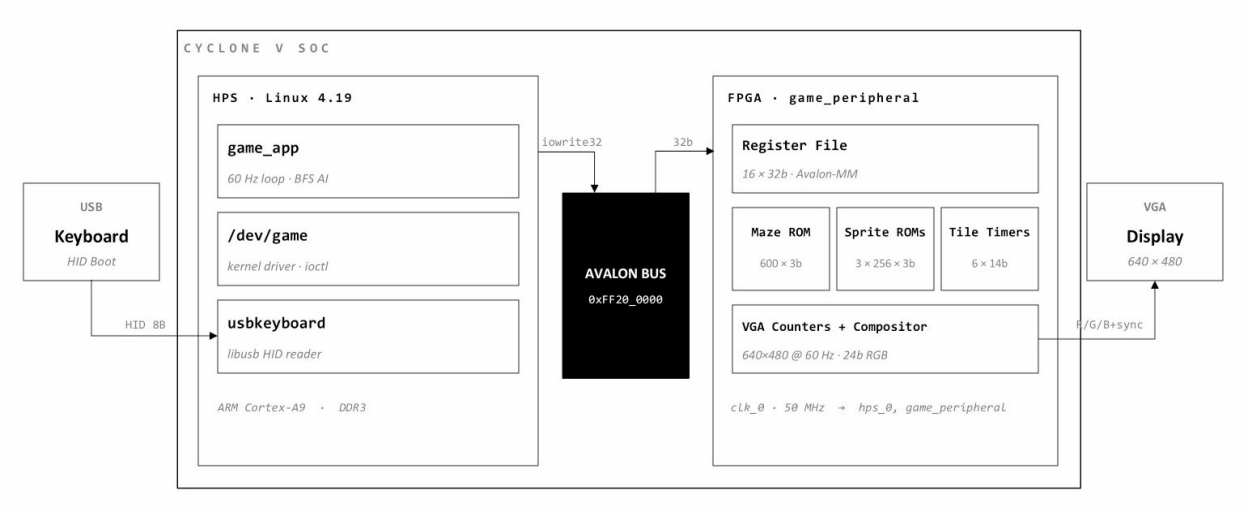

Figure 1: Top-level system block diagram. The HPS (left, inside the Cyclone V SoC) runs Linux 4.19; `game_app` drives the loop, `/dev/game` is the kernel driver, and `usbkeyboard` consumes the 8-byte HID Boot reports coming off the on-board USB controller. The FPGA side (right) holds the custom `game_peripheral` with its register file, ROMs, six tile timers, and the VGA counters + compositor that drive the off-board VGA display. The Avalon bus at `0xFF20_0000` is the only path between the two sides.

2.2 Inside the game peripheral

Everything inside `game_peripheral.sv` lives in one file, mirroring the lab3 `vga_ball.sv` pattern. Six modules cooperate:

Sub-module	Role
vga_counters	640 × 480 / 60 Hz `hcount`/`vcount`; drives `VGA_HS/VS/BLANK/SYNC`; emits a 1-cycle `vsync_pulse` at frame start.
vsync_cdc	2-FF synchroniser + rising-edge detect that brings `vsync_pulse` into the clk domain as `vsync_tick`.
tile_timer (× 6)	14-bit saturating counter for each Special Tile; counts `vsync_ticks` while the Evader is on that tile and asserts `activated` once the count reaches 180.
maze_rom	600-word × 3-bit ROM (M10K) holding the cell type for every `(col,row)`.
sprite_rom (× 3)	256-word × 3-bit ROM, one per entity, addressed by `(py_in_tile, px_in_tile)`.
compositor	Combinational mux: per pixel, picks the final 24-bit RGB from the sprite layer, the tile layer, the gate/activation state, or the win/loss overlay.

2.3 The two domains' contract

The HPS owns: where the Evader and ghosts are, when the gate is open, and the win/loss bits. The FPGA owns: which pixel is being drawn right now, which Special Tile (if any) currently has the Evader standing on it, and the rolling 16-bit frame counter that paces the HPS. The register file is the only conduit and is described in detail in Section 5.

3 Hardware Design (FPGA)

3.1 Screen layout

The VGA frame is 640 × 480. The maze is 30 tiles wide × 20 tiles tall at 16 × 16 pixels each, occupying 480 × 320 pixels and centered inside the frame with 80-pixel black borders.

`vga_counters` uses an 11-bit `hcount` whose top 10 bits give the pixel column (so `hcount[10:1]` is the 0–639 pixel index and `hcount[0]` is the 25 MHz `VGA_CLK` strobe).

```
logic [9:0] pixel_col = hcount[10:1];          // 0..639
logic [9:0] pixel_row = vcount;                // 0..479

logic in_maze = (pixel_col >= 80) && (pixel_col < 560)
             && (pixel_row >= 80) && (pixel_row < 400);

logic [9:0] maze_x = pixel_col - 80;           // 0..479
logic [9:0] maze_y = pixel_row - 80;           // 0..319

assign tile_col   = maze_x[8:4];               // maze_x / 16
assign tile_row   = maze_y[8:4];
assign px_in_tile = maze_x[3:0];               // maze_x % 16
assign py_in_tile = maze_y[3:0];

assign maze_addr  = tile_row * 30 + tile_col;  // row-major index
assign sprite_addr = {py_in_tile, px_in_tile};
```

3.2 Fixed-tile positions

Tile	Col	Row	Screen X	Screen Y
Special Tile 0	3	2	128	112
Special Tile 1	26	2	496	112
Special Tile 2	3	10	128	240
Special Tile 3	26	10	496	240
Special Tile 4	14	17	304	352
Special Tile 5	17	17	352	352
Gate Tile	15	1	320	96

3.3 Tile activation

Each Special Tile has its own `tile_timer` parameterised by `TILE_COL` and `TILE_ROW`. On every `vsync_tick`, the timer checks whether the Evader's register position matches its hard-coded tile, and if so, increments a 14-bit counter. Once the count reaches **180** (≈ 3 s at 60 Hz) the `activated` output goes high and stays high until the next reset.

The threshold was originally 600 frames (10 s) in the proposal. We dropped it to 180 during integration because the demo felt sluggish; three seconds per tile is enough that the player has to commit, but quick enough that an entire play session is over within a minute or two.

3.4 Compositor priority

The compositor is purely combinational from the registered ROM outputs down to the 24-bit RGB output, with the following priority:

1. If `!VGA_BLANK_n` the output is black (the DAC requires this during blanking).
2. If `evader_win` or `ghost_win` is set, the entire screen is painted green or red respectively. The win/loss bits are written by the HPS into GAME_STATUS and persist until `GAME_RESET`.
3. If `!in_maze` the pixel is part of the outer border and renders black.
4. Sprite layer: if the current tile contains an entity and that entity's sprite pixel is non-zero (non-transparent), the corresponding palette colour is used. Priority within a tile is Evader ¿ Ghost 0 ¿ Ghost 1.
5. Otherwise the pixel is part of the maze tile layer: `maze_cell` drives the tile colour, modulated by `tile_activated[]` (green when active, dim yellow when inactive) and `gate_open` (green if open, grey if locked).

3.5 Memory layout

Memory	Contents	Size	M10K blocks
Maze ROM	600 cells × 3 bits	1,800 bits	1
Sprite (Evader)	16 × 16 × 3 b	768 bits	1
Sprite (Ghost 0)	16 × 16 × 3 b	768 bits	1
Sprite (Ghost 1)	16 × 16 × 3 b	768 bits	1
Total (upper bound)		4,104 bits	4
After Quartus sharing			**2**

The Quartus Fitter report shows that the four ROMs end up packed into just 2 M10K blocks total. We did not have to ask for this explicitly; it is the natural result of how Quartus consolidates small ROMs whose ports don't conflict.

4 Software Design (HPS)

The HPS side is split between a small Linux kernel module (`game.ko`) and a user-space game application (`game_app`).

4.1 Kernel module

`game.c` is a platform driver. It registers a misc device at `/dev/game`, locates the peripheral through the device tree (compatible string `"csee4840,game_peripheral-1.0"`, generated by `sopc2dts` into the `.dts`/`.dtb`), maps the 64-byte BAR with `of_iomap`, and routes four ioctls to `iowrite32`/`ioread32` calls.

ioctl	Behaviour
`GAME_WRITE_STATE`	Pushes new Evader / Ghost positions to the FPGA. Optionally also writes GAME_STATUS (the per-call `write_status` flag controls this).
`GAME_READ_STATE`	Returns TILE_STATUS, all six TILE*_TIME counters, GAME_STATUS, and the frame counter in a single struct.
`GAME_RESET_GAME`	Writes GAME_RESET into GAME_STATUS, pulsing the self-clearing reset bit.
`GAME_WAIT_VSYNC`	Busy-polls FRAME_SYNC inside the driver until FRAME_COUNT advances. Returns the new value. `cond_resched()` every ∼4 K spins keeps the kernel preempt-friendly.

We chose the kernel-driver + ioctl path over `mmap /dev/mem` because (a) the lab3 reference does it that way, so we knew the boilerplate worked, and (b) `GAME_WAIT_VSYNC` is a clean place to hide the busy-poll: user code calls one ioctl and blocks; it doesn't have to know how the hardware signals frame transitions.

4.2 Main game loop

The loop in `game_app.c` runs at 60 Hz, paced by `GAME_WAIT_VSYNC`. Entity movement happens at 5 Hz (every 12 frames); ghost AI runs on every second movement tick, so ghosts are exactly half the Evader's speed.

```
int move_tick = 0, ghost_tick = 0;
while (!peek_esc()) {
    wait_vsync();                       /* blocks ~16.7 ms */
    poll_input();                       /* sample the kbd thread's state */

    if (g_game_over) {
        if (consume_enter()) reset_game_hw();
        continue;
    }

    game_read_t st; read_state(&st);    /* tile_status, etc.   */
    check_gate_unlock(st.tile_status);  /* opens gate at popcount>=3 */

    if (++move_tick >= FRAMES_PER_TICK) {   /* every 12 frames = 5 Hz */
```

```
        move_tick = 0;
        move_evader();
        if (++ghost_tick >= GHOST_TICKS_PER_MOVE) {
            ghost_tick = 0;
            move_ghosts();              /* BFS for both ghosts */
        }
        send_state(0, 0);               /* push positions only */
        check_win_loss();
    }
}
```

4.3 Ghost AI

Each ghost runs an independent BFS over the 600-cell grid on every movement tick. The workspace is in `.bss`, so no heap allocation happens at runtime:

```
#define BFS_N (GAME_MAZE_ROWS * GAME_MAZE_COLS)   /* = 600 */
static uint8_t  bfs_visited[BFS_N];
static int16_t  bfs_parent [BFS_N];
static uint16_t bfs_queue  [BFS_N];
```

BFS guarantees the shortest path in $O(N) = O(600)$ operations. After it reaches the Evader's cell, we walk the `parent[]` array backwards until we find the immediate neighbour of the ghost's current cell, and that neighbour is the next move. Profiling a release build on the A9 showed each BFS call completing in roughly 4–10 μs, so two calls per ghost tick at 5 Hz is well under 1/1000 of CPU.

One subtle point: ghosts are not allowed to walk through the gate cell even when the gate is open. `maze_passable_ghost()` returns 0 for both walls and the gate, so the ghost BFS treats the gate as a wall. Otherwise the ghosts would happily run through the gate as a shortcut and the player could never use it as a winning destination.

4.4 USB keyboard reader

`usbkeyboard.c` walks the libusb device list looking for the first interface whose class is HID (0x03) and whose protocol byte is Boot Keyboard (0x01). When it finds one, it detaches the kernel driver if needed, claims the interface, and returns the device handle plus the IN endpoint address.

`game_app.c` spawns one pthread that loops on `libusb_interrupt_transfer` with a 200 ms timeout, reading the standard 8-byte HID Boot keyboard report. Inside the loop:

- Each report contains up to six simultaneously held keycodes (HID Usage Page 0x07).
- We scan for the four arrow IDs (0x4F-0x52) and write the first match into `g_held_dir` under a mutex.
- ENTER (0x28) and ESC (0x29) are edge-triggered: the flag goes high only when the key transitions from not-pressed to pressed in two consecutive reports.

Why this and not `/dev/input/eventX`? On the lab kernel, `usbhid` matches the keyboard's Boot interface for arrows but its Consumer interface for media keys, and the input-layer device numbering wasn't deterministic across reboots. Pulling the HID Boot report directly with libusb sidestepped all of that.

5 Hardware/Software Interface

Base physical address: `0xFF20_0000` (Lightweight HPS-to-FPGA bridge). The peripheral exposes 16 word-addressable 32-bit registers, of which 12 are actually used. `avalon_slave_0.address` is 4 bits wide with `addressUnits = WORDS`.

5.1 Register map

Offset	Word	Name	Acc.	Written by	Description
0x00	0	EVADER_POS	R/W	HPS	Evader tile position
0x04	1	GHOST0_POS	R/W	HPS	Ghost 0 tile position
0x08	2	GHOST1_POS	R/W	HPS	Ghost 1 tile position
0x0C	3	TILE_STATUS	R	FPGA	[5:0] activation flags
0x10	4	TILE0_TIME	R	FPGA	Frame count, tile 0
0x14	5	TILE1_TIME	R	FPGA	Frame count, tile 1
0x18	6	TILE2_TIME	R	FPGA	Frame count, tile 2
0x1C	7	TILE3_TIME	R	FPGA	Frame count, tile 3
0x20	8	TILE4_TIME	R	FPGA	Frame count, tile 4
0x24	9	TILE5_TIME	R	FPGA	Frame count, tile 5
0x28	10	GAME_STATUS	R/W	HPS+FPGA	See bit map below
0x2C	11	FRAME_SYNC	R	FPGA	16-bit frame counter, vsync flag
0x30--0x3C	12–15	reserved	—	—	reads 0, writes ignored

5.2 Position register bit layout

EVADER_POS, GHOST0_POS, GHOST1_POS.

Bits	Field	Description
[4:0]	COL	Tile column 0–29. Five bits is enough ($2^5 = 32 \geq 30$).
[9:5]	ROW	Tile row 0–19.
[31:10]	—	Unused; reads back 0.

5.3 TILE_STATUS bit layout

Bits	Name	Description
[0]	TILE0_ACT	1 once Special Tile 0's count reaches 180.
[1]	TILE1_ACT	
[2]	TILE2_ACT	
[3]	TILE3_ACT	
[4]	TILE4_ACT	
[5]	TILE5_ACT	
[31:6]	—	Unused; reads back 0.

5.4 GAME_STATUS bit layout

Bits	Name	Description
[0]	GATE_OPEN	HPS sets this once any three TILE*_ACT bits are high.
[1]	EVADER_WIN	HPS sets this when the Evader stands on the open Gate. The FPGA paints the maze green.
[2]	GHOST_WIN	HPS sets this when a ghost collides with the Evader. The FPGA paints the maze red.
[3]	GAME_RESET	Self-clearing. HPS writes 1 to reset entity positions and clear the win/loss bits; the hardware clears bit 3 on the cycle it sees it asserted.
[31:4]	—	Unused; reads back 0.

5.5 FRAME_SYNC bit layout

Bits	Field	Description
[15:0]	FRAME_COUNT	16-bit rolling counter; wraps every $2^{16}/60 \approx 1{,}092$ s ($\sim$18 minutes).
[16]	VSYNC_TICK	Reflects the synchronised `vsync_tick`. Reads 1 only during the one clk cycle in which the tick is asserted; useful for waveform debugging only.
[31:17]	—	Unused; reads back 0.

5.6 Avalon-MM protocol

The slave is a single-cycle response slave; `readWaitTime` is 1 and `writeWaitTime` is 0. The HPS bridge therefore issues a read, sees the data one cycle later, and we don't need a `waitrequest` signal. All four protocol signals (`chipselect`, `read`, `write`, `address`) come from the Qsys-generated lightweight master.

6 Build and Boot Flow

6.1 Hardware build (on the workstation, with Quartus on PATH)

```
cd final_project-hw
python3 gen_mif.py     # writes *.mif, *.hex, and ../final_project-sw/maze_data.h
make qsys              # qsys-generate
make project           # quartus_sh -t soc_system.tcl + map + HPS pin TCL
make quartus           # full compile -> output_files/soc_system.sof
make rbf               # .sof -> .rbf for SD-card boot
make dtb               # sopc2dts + dtc -> soc_system.dtb
```

`rbf` and `dtb` go onto the FAT partition of the SD card alongside the lab2 Linux image. We did **not** rebuild U-Boot, the kernel, or the preloader; the lab2 image already contains everything needed.

6.2 Software build (on the DE1-SoC, after booting from that SD card)

```
cd final_project-sw
make                   # builds game.ko and game_app
insmod game.ko         # probes the device at 0xFF20_0000, creates /dev/game
./game_app             # finds and claims the USB keyboard, runs the game
```

A USB HID keyboard must be plugged into the on-board USB-A port. We tested with two Logitech keyboards (a K360 and a generic wired model) and a Dell unbranded keyboard; all three were enumerated correctly.

6.3 Makefile changes vs. lab3-hw

For reference, our hardware Makefile is structurally similar to lab3-hw, with these additions / removals:

Added

- New `ROM_FILES` variable listing the eight ROM init files plus `maze_data.h`.
- `.PHONY: roms`, with `python3 gen_mif.py` as the recipe.
- `make qsys` now depends on `$(ROM_FILES)` so that `qsys-generate` picks up freshly built ROMs.
- `rom-clean` for housekeeping.

Removed

- Preloader / U-Boot / kernel-build targets (we reuse the lab2 SD-card image).
- The associated BSP / KERNEL / CROSS variables and their clean targets.

7 Lessons Learned and Things That Surprised Us

7.1 The proposal was bigger than the project we should have proposed

The original proposal sketched a more ambitious system with multiple play modes, a hardware-accelerated pursuit-evaluation pipeline, audio output via the WM8731 codec, SD-card map loading, and a USB gamepad. That was clearly too much for the time budget. We dropped audio first, then SD-card maps, then the gamepad path, then finally we collapsed the play modes down to "Human vs. AI" and put BFS on the HPS where it naturally belonged. *Almost every cut made the system better, not just smaller*: fewer moving parts, fewer integration points, fewer ways for the demo to fail in front of the TA.

The lesson is that the proposal stage is the point at which you most overestimate what you can finish, and the moments where you find yourself reaching for an extra clock domain or an extra peripheral are usually the moments where you should stop and ask whether the basic system is actually working yet.

7.2 Lab3 is the reference design

By the time we got to integration, the things that worked instantly were the things we kept from lab3 (the 50 MHz clocking pattern, the active-high reset, the misc-device + ioctl driver, the unused-peripheral squelch block in the top wrapper). The things that took the longest to debug were the things we tried to do differently. Our first design wanted a dedicated 25.175 MHz PLL and active-low reset, which together would have introduced one new clock domain and one new reset polarity for the same visible behaviour.

If we did the project again, we would treat the lab3 starter as a hard template and only deviate when there's a concrete reason. "Cleaner" is not a reason.

7.3 The HID composite-device trap

This one cost us the most hours. The keyboard we plugged in worked perfectly with the Linux `usbhid` driver in the sense that it showed up under `/dev/input/`, but the device numbering was non-deterministic across reboots (sometimes `event0`, sometimes `event1`), and arrow keys came from one subdevice while ENTER came from another. After some time trying to filter and merge events the right way, we backed up and lifted the libusb HID-Boot reader from lab2 instead. That now reliably gets arrows + ENTER + ESC from one source.

A useful piece of meta-advice: if a stack is unreliable, look for the lowest-level interface you can read *at the same speed*. We needed 60 Hz responsiveness; the USB interrupt endpoint polls every 8 ms; libusb is right there.

7.4 CDC is cheap insurance

Even though our `vsync_pulse` and `vsync_tick` are technically in the same clock domain (both driven from the 50 MHz net), we kept the 2-FF synchroniser and edge detector. This came from a TA's comment during design review: "Treat any pulse that crosses between two counters as a CDC pulse, whether or not the clocks happen to be the same right now." The day we (or someone

reading our code in 2027) introduce a separate pixel-clock PLL, the synchroniser is already there. The cost in area and timing is negligible.

7.5 Tile counters belong on the FPGA

A simple thing that mattered: per-tile activation counts have to be exact. The first prototype counted frames in user space (decrementing a per-tile timer based on the held position) and it felt floppy — the count drifted under Linux load, because the user-space loop missed an occasional vsync when the kernel had something else to do. Pushing the counter into the FPGA and reading it back made the gameplay feel solid and predictable. This is the sort of thing the FPGA is for.

7.6 What we wish we had known earlier

- `quartus_pgm -m jtag` to load a fresh `.sof` after each iteration is faster than rewriting the SD card every time. We didn't discover this until two weeks in.
- `$readmemh` accepts one hex digit per line, which means our 3-bit ROMs just write one nibble per address. The Quartus `.mif` parser, on the other hand, is fussier; that's why `gen_mif.py` emits both formats.
- When you instantiate a parameterised module multiple times with different parameters, the `(* ramstyle = "M10K" *)` hint travels with the instance, not the module — Quartus is willing to pack several small parameterised ROMs into one M10K block on its own.

8 Division of Work

The team is four people and the project naturally splits into four ribbons of work:

Member	Responsibilities
Ryan Lo (yl5972)	VGA rendering pipeline. Wrote `vga_counters`, `vsync_cdc`, and the compositor inside `game_peripheral.sv`. Designed the screen layout, the eight-colour palette, and the per-pixel priority logic. Did the early prototype on the lab3 vga_ball reference and was responsible for everything pixel-clock related (including the 50 MHz / hcount[0] strobe decision). Authored Sections 2–3 and 7 of this report.
Junhao Qu (jq2434)	Maze ROM and tile activation. Wrote `maze_rom`, `sprite_rom`, and the six `tile_timer` instances. Owned `gen_mif.py` and the maze-layout/connectivity invariants. Tuned the 180-frame threshold and the popcount ≥ 3 gate unlock rule. Verified the `maze_data.h` round-trip between Python and the C code.
Boxiong Li (bl3155)	Qsys integration and build flow. Wrote `soc_system_top.sv`, `game_peripheral_hw.tcl`, and the hardware Makefile. Handled Quartus project bring-up (pin assignments, HPS pin TCL, .qpf/.qsf/.sdc), `.rbf` and `.dtb` generation, and the SD-card boot procedure. Drove the design-review demo and most of the in-lab debugging sessions.
Kevin Liu (kl3755)	HPS software. Wrote the Linux kernel module (`game.c`/`game.h`) and the user-space game application (`game_app.c`), including the 60 Hz main loop, the win/loss FSM, and the BFS ghost AI. Implemented the USB keyboard reader on top of the lab2 libusb harness and built the pthread + mutex plumbing connecting the keyboard thread to the main loop.

In practice we paired heavily across these ribbons: Ryan and Junhao integrated the compositor with the ROMs together, Boxiong and Kevin debugged the device-tree binding for `/dev/game`, and everyone debugged Quartus warnings at one point or another. The git log shows the cleanest split through the design-review milestone; afterwards the commits get smaller and more cross-cutting as we were tracking down integration issues.

9 Advice for Future Projects

1. **Propose 60% of what you think you can do.** The other 40% will appear in the form of integration overhead, Quartus surprises, and one of the labs that nobody on the team finished cleanly. Save audio and a second peripheral for "stretch."

2. **Build the simplest end-to-end thing first.** Our turning point was the day we had a static maze on the screen, a single Evader sprite that responded to arrow keys, and a fake ghost that moved in a straight line. Everything afterwards was incremental.

3. **Pin down the interface contract before either side is finished.** We wrote the register map into `game.h` and `game_peripheral.sv` at the same time, so both sides could be developed in parallel without merge fights. Once the bit layout was fixed, the rest of the work fell out.

4. **Stay close to lab3.** The toolchain, the device tree, the Makefile, the misc-device pattern — all of these are baked into the course infrastructure for a reason. Deviating from them means you spend hours debugging the deviation, not the project.

5. **Treat the FPGA as the timing source.** Anything that needs to be exactly periodic — 60 Hz frame pacing, per-tile counters, blink rates, audio sample clocks — should be on the FPGA. The HPS reads back; it doesn't drive timing.

6. **Write a `gen_*.py` script for any artefact that gets compiled into two languages.** Ours generates the `.mif`, `.hex`, and `maze_data.h` files all from the same source. If you make the FPGA and the HPS disagree about what the maze is, you'll spend hours debugging gameplay before you realise the bug is in the data.

7. **Demo the project on the actual board well before the deadline.** The first time we tried to run `game_app` on the DE1-SoC instead of just simulating, we found two race conditions and a wrong-polarity reset within an hour. The earlier you find these, the easier they are.

A Source Code Listings

This appendix contains every file we wrote by hand for the project. Files generated automatically by Quartus, Qsys, or the kernel build system (`soc_system.sopcinfo`, `soc_system.qpf`, `soc_system.qsf`, `soc_system.sdc`, output_files/*, etc.) are intentionally omitted, as are the auto-generated `maze_data.h`, `*.mif`, and `*.hex` files (produced from `gen_mif.py`, which is included).

A.1 Project README (README.md)

```
# Maze Chase Game

CSEE 4840 Embedded Systems · Spring 2026 · Final Project

Ryan Lo (yl5972) · Junhao Qu (jq2434) · Boxiong Li (bl3155) · Kevin Liu (kl3755)

A real-time maze chase game running on the Terasic DE1-SoC (Cyclone V FPGA +
ARM HPS under Linux). The Evader runs around a fixed 30×20 maze trying to
charge up enough special tiles to unlock a gate, while two ghosts close in on
it using BFS pathfinding. Hardware draws the VGA frame; software runs the game.

‘‘‘
4840_final-main/
+-- final_project-hw/   FPGA: SystemVerilog peripheral, Qsys system, ROMs
+-- final_project-sw/   HPS: kernel module + user-space game application
‘‘‘

The directory layout deliberately mirrors ‘lab3/‘ so the toolchain works the
same way.

## What’s in each tree

### ‘final_project-hw/‘

| File | Purpose |
|---|---|
| ‘game_peripheral.sv‘ | The custom Avalon-MM peripheral. Contains the top module plus ‘vga_counters‘, ‘vsync_cdc‘, ‘tile_timer‘, ‘maze_rom‘, ‘sprite_rom‘, and ‘compositor‘. |
| ‘soc_system_top.sv‘ | Board-level wrapper. Instantiates the Qsys-generated ‘soc_system‘ and connects the VGA conduit to the off-chip ADV7123 DAC. |
| ‘soc_system.qsys‘ | Qsys system with ‘clk_0‘, ‘hps_0‘, and ‘game_peripheral_0‘. |
| ‘game_peripheral_hw.tcl‘ | Component descriptor that lets Qsys see our peripheral. |
| ‘gen_mif.py‘ | Python script that emits the four ROM init files (‘.mif‘ + ‘.hex‘) and a matching ‘maze_data.h‘ for the HPS side. |
| ‘soc_system.tcl‘ | Quartus project bring-up (pin assignments, top entity, etc.). Adapted from the lab3 starter. |
| ‘soc_system_board_info.xml‘ | Needed by ‘sopc2dts‘ when generating the device tree. |
| ‘Makefile‘ | Build flow: ‘roms → qsys → project → quartus → rbf → dtb‘. |

### ‘final_project-sw/‘

| File | Purpose |
|---|---|
```

```
| `game.c` / `game.h` | Linux kernel module that exposes the peripheral as `/dev/game` and translates four ioctls into 32-bit Avalon writes/reads. |
| `game_app.c` | Userspace game loop. 60 Hz frame pacing, BFS ghost AI, USB-keyboard pthread, win/loss FSM. |
| `usbkeyboard.c` / `usbkeyboard.h` | libusb wrapper that opens the first HID-Boot-Keyboard endpoint we can find. Lifted from lab2 and lightly extended. |
| `maze_data.h` | Auto-generated by `gen_mif.py`. Mirrors the maze ROM so the HPS sees the same walls the FPGA does. |
| `Makefile` | Builds `game.ko` and `game_app`. |
| `README` | Build & run notes specific to the HPS side. |

## Build

### Hardware (development workstation, with Quartus on PATH)

```sh
cd final_project-hw
python3 gen_mif.py     # writes *.mif, *.hex, and ../final_project-sw/maze_data.h
make qsys              # qsys-generate
make project           # quartus_sh -t soc_system.tcl + map + HPS pin assignments
make quartus           # full compile -> output_files/soc_system.sof
make rbf               # .sof -> .rbf for SD-card boot
make dtb               # sopc2dts + dtc -> soc_system.dtb
```

`rbf` and `dtb` go onto the FAT partition of the SD card alongside the lab2
Linux image.

Software (on the DE1-SoC, after booting from that SD card)

```sh
cd final_project-sw
make                   # builds game.ko + game_app
insmod game.ko
./game_app
```

Play

* Arrow keys move the Evader.
* Stand on three of the six special tiles long enough to light them up -- the
  gate at the top of the maze unlocks once any three are activated.
* Reach the gate to win. Get caught by a ghost to lose.
* `ENTER` restarts after win/loss. `ESC` quits.

Notes

* The `.hex` files exist in addition to the `.mif` files because we drive the
  ROMs with `$readmemh`, which the simulator we used for early test benches
  understands and the `.mif` reader doesn't. The two formats encode the
  identical maze and sprite data.
* `gen_mif.py` writes `maze_data.h` to **both** trees. Quartus and the kernel
  module both see the same maze without any manual copying.
* The ghosts use BFS in software, not in hardware -- BFS over 600 nodes is
  a few microseconds on the A9 and Linux can comfortably keep up at 5 Hz.
  Putting it on the FPGA would have been overkill for this size of maze.
* `vsync_cdc` is the one place we had to be careful about clock-domain
  crossing. The VGA counter runs off the 50 MHz board clock but pulses once
```

```
94    per frame, so we synchronize that pulse before it drives the 6 tile timers
95    and the frame counter.
```

A.2 Hardware: game_peripheral.sv

```
// Maze Chase Game - custom Avalon-MM peripheral.
// Wraps a 640x480 VGA generator and a small register file that the HPS
// reads and writes over the Lightweight HPS-to-FPGA bridge.  Structure
// follows lab3's vga_ball.sv: this one file holds the top module and
// every sub-module the design uses.
//
// Register map (byte offsets, 32-bit words):
//   0x00 EVADER_POS   R/W (HPS)  {row[4:0], col[4:0]}
//   0x04 GHOST0_POS   R/W (HPS)
//   0x08 GHOST1_POS   R/W (HPS)
//   0x0C TILE_STATUS  R   (FPGA) [5:0] activation flags
//   0x10 TILE0_TIME   R   (FPGA) [13:0] frames spent on tile 0
//   0x14 TILE1_TIME   R   (FPGA)
//   0x18 TILE2_TIME   R   (FPGA)
//   0x1C TILE3_TIME   R   (FPGA)
//   0x20 TILE4_TIME   R   (FPGA)
//   0x24 TILE5_TIME   R   (FPGA)
//   0x28 GAME_STATUS  R/W (HPS)  [0] gate_open  [1] ev_win
//                                [2] gh_win     [3] reset (self-clearing)
//   0x2C FRAME_SYNC   R   (FPGA) [15:0] frame#  [16] vsync_tick
//   0x30..0x3C        reserved (reads 0, writes ignored)

module game_peripheral(
    input  logic        clk,            // 50 MHz system clock
    input  logic        reset,          // active-high synchronous reset

    // Avalon-MM slave (Lightweight HPS-to-FPGA bridge)
    input  logic [3:0]  address,        // word address (16 regs)
    input  logic        chipselect,
    input  logic        write,
    input  logic        read,
    input  logic [31:0] writedata,
    output logic [31:0] readdata,

    // VGA outputs (direct to DE1-SoC ADV7123)
    output logic [7:0]  VGA_R, VGA_G, VGA_B,
    output logic        VGA_CLK,
    output logic        VGA_HS,
    output logic        VGA_VS,
    output logic        VGA_BLANK_n,
    output logic        VGA_SYNC_n
);

    // ------------------------------------------------------------------
    // Internal nets
    // ------------------------------------------------------------------
    logic [10:0] hcount;                // hcount[10:1] is pixel column
    logic [9:0]  vcount;
    logic        vsync_pulse;           // pixel_clk-domain 1-cycle pulse
    logic        vsync_tick;            // clk-domain CDC-safe pulse
    logic        pixel_clk;

    // Register file outputs
    logic [4:0]  evader_col, evader_row;
    logic [4:0]  ghost0_col, ghost0_row;
    logic [4:0]  ghost1_col, ghost1_row;
```

```
    logic [3:0]  game_status_q;          // {reset, gh_win, ev_win, gate_open}

    // Tile timers
    logic [5:0]  tile_activated;
    logic [13:0] tile_count [0:5];

    // Frame counter
    logic [15:0] frame_count;

    // Maze / sprite ROM outputs
    logic [9:0]  maze_addr;
    logic [2:0]  maze_cell;
    logic [7:0]  sprite_addr;
    logic [2:0]  evader_pixel, ghost0_pixel, ghost1_pixel;

    // ------------------------------------------------------------------
    // VGA timing -> hcount/vcount + VGA control signals
    // ------------------------------------------------------------------
    vga_counters u_counters (
        .clk50       (clk),
        .reset       (reset),
        .hcount      (hcount),
        .vcount      (vcount),
        .VGA_CLK     (VGA_CLK),
        .VGA_HS      (VGA_HS),
        .VGA_VS      (VGA_VS),
        .VGA_BLANK_n (VGA_BLANK_n),
        .VGA_SYNC_n  (VGA_SYNC_n),
        .vsync_pulse (vsync_pulse)
    );

    // pixel_clk derives from hcount[0] -> 25 MHz strobe; ROMs use clk and
    // sample every other cycle.  We use clk as the ROM clock and gate the
    // address with hcount[0] so the registered output appears on the right
    // cycle.  Simpler: just use clk; address changes once per pixel pair.
    assign pixel_clk = clk;  // unused but kept for clarity

    // ------------------------------------------------------------------
    // CDC: bring vsync from pixel-clock counter to system clock
    // ------------------------------------------------------------------
    vsync_cdc u_cdc (
        .clk         (clk),
        .reset       (reset),
        .vsync_pulse (vsync_pulse),
        .vsync_tick  (vsync_tick)
    );

    // Maintain a 16-bit frame counter incremented on every vsync_tick
    always_ff @(posedge clk) begin
        if (reset)
            frame_count <= 16'd0;
        else if (vsync_tick)
            frame_count <= frame_count + 16'd1;
    end

    // ------------------------------------------------------------------
    // Avalon-MM register file
    // ------------------------------------------------------------------
    // Bit layout for *_POS regs: writedata[4:0]=col, writedata[9:5]=row
```

```
    logic do_write;
    logic [9:0] evader_pos_q, ghost0_pos_q, ghost1_pos_q;

    assign do_write = chipselect && write;
    assign evader_col = evader_pos_q[4:0];
    assign evader_row = evader_pos_q[9:5];
    assign ghost0_col = ghost0_pos_q[4:0];
    assign ghost0_row = ghost0_pos_q[9:5];
    assign ghost1_col = ghost1_pos_q[4:0];
    assign ghost1_row = ghost1_pos_q[9:5];

    // GAME_RESET is self-clearing: hardware clears bit[3] on the cycle it
    // sees it asserted.  Win/loss/gate flags are sticky until reset.
    always_ff @(posedge clk) begin
        if (reset) begin
            // Initial entity positions (must all be passable cells)
            evader_pos_q  <= {5'd10, 5'd15};  // (col=15, row=10) center
            ghost0_pos_q  <= {5'd2,  5'd1 };  // (col=1,  row=2)
            ghost1_pos_q  <= {5'd2,  5'd28};  // (col=28, row=2)
            game_status_q <= 4'b0000;
        end else begin
            if (game_status_q[3]) begin
                // GAME_RESET asserted: clear everything
                evader_pos_q  <= {5'd10, 5'd15};
                ghost0_pos_q  <= {5'd2,  5'd1 };
                ghost1_pos_q  <= {5'd2,  5'd28};
                game_status_q <= 4'b0000;
            end else if (do_write) begin
                case (address)
                    4'd0 : evader_pos_q  <= writedata[9:0];
                    4'd1 : ghost0_pos_q  <= writedata[9:0];
                    4'd2 : ghost1_pos_q  <= writedata[9:0];
                    4'd10: game_status_q <= writedata[3:0];
                    default: /* read-only or reserved */;
                endcase
            end
        end
    end

    // Read-mux
    always_comb begin
        case (address)
            4'd0 : readdata = {22'd0, evader_pos_q};
            4'd1 : readdata = {22'd0, ghost0_pos_q};
            4'd2 : readdata = {22'd0, ghost1_pos_q};
            4'd3 : readdata = {26'd0, tile_activated};
            4'd4 : readdata = {18'd0, tile_count[0]};
            4'd5 : readdata = {18'd0, tile_count[1]};
            4'd6 : readdata = {18'd0, tile_count[2]};
            4'd7 : readdata = {18'd0, tile_count[3]};
            4'd8 : readdata = {18'd0, tile_count[4]};
            4'd9 : readdata = {18'd0, tile_count[5]};
            4'd10: readdata = {28'd0, game_status_q};
            4'd11: readdata = {15'd0, vsync_tick, frame_count};
            default: readdata = 32'd0;
        endcase
    end

    // ------------------------------------------------------------------
```

```
    // Tile timers (6 instances at hardcoded grid positions)
    // ------------------------------------------------------------------
    // Hardcoded special-tile positions from the design document
    localparam logic [4:0] T0_COL = 5'd3,  T0_ROW = 5'd2;
    localparam logic [4:0] T1_COL = 5'd26, T1_ROW = 5'd2;
    localparam logic [4:0] T2_COL = 5'd3,  T2_ROW = 5'd10;
    localparam logic [4:0] T3_COL = 5'd26, T3_ROW = 5'd10;
    localparam logic [4:0] T4_COL = 5'd14, T4_ROW = 5'd17;
    localparam logic [4:0] T5_COL = 5'd17, T5_ROW = 5'd17;
    localparam logic [4:0] GATE_COL = 5'd15, GATE_ROW = 5'd1;

    // Single combined reset: external reset OR self-clearing GAME_RESET
    logic reset_int;
    assign reset_int = reset | game_status_q[3];

    tile_timer #(.TILE_COL(T0_COL), .TILE_ROW(T0_ROW)) u_t0 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[0]), .activated(tile_activated[0]));
    tile_timer #(.TILE_COL(T1_COL), .TILE_ROW(T1_ROW)) u_t1 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[1]), .activated(tile_activated[1]));
    tile_timer #(.TILE_COL(T2_COL), .TILE_ROW(T2_ROW)) u_t2 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[2]), .activated(tile_activated[2]));
    tile_timer #(.TILE_COL(T3_COL), .TILE_ROW(T3_ROW)) u_t3 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[3]), .activated(tile_activated[3]));
    tile_timer #(.TILE_COL(T4_COL), .TILE_ROW(T4_ROW)) u_t4 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[4]), .activated(tile_activated[4]));
    tile_timer #(.TILE_COL(T5_COL), .TILE_ROW(T5_ROW)) u_t5 (
        .clk(clk), .reset(reset_int), .vsync_tick(vsync_tick),
        .evader_col(evader_col), .evader_row(evader_row),
        .count(tile_count[5]), .activated(tile_activated[5]));

    // ------------------------------------------------------------------
    // Maze rendering coordinates (pixel-clock domain operates on clk too,
    // since vga_counters uses clk50 to produce 25 MHz strobe via hcount[0])
    // ------------------------------------------------------------------
    logic [9:0] pixel_col, pixel_row;
    assign pixel_col = hcount[10:1];          // 0..639
    assign pixel_row = vcount;                // 0..479

    logic in_maze_x, in_maze_y, in_maze;
    assign in_maze_x = (pixel_col >= 10'd80) && (pixel_col < 10'd560);
    assign in_maze_y = (pixel_row >= 10'd80) && (pixel_row < 10'd400);
    assign in_maze   = in_maze_x && in_maze_y;

    logic [9:0]  maze_x, maze_y;
    assign maze_x = pixel_col - 10'd80;       // 0..479
    assign maze_y = pixel_row - 10'd80;       // 0..319

    logic [4:0]  tile_col, tile_row;
    logic [3:0]  px_in_tile, py_in_tile;
```

```
    assign tile_col   = maze_x[8:4];
    assign tile_row   = maze_y[8:4];
    assign px_in_tile = maze_x[3:0];
    assign py_in_tile = maze_y[3:0];

    // Tile-index address: row*30 + col
    // Multiplication by 30 = (row<<5) - (row<<1)
    assign maze_addr   = ({5'd0, tile_row} * 10'd30) + {5'd0, tile_col};
    assign sprite_addr = {py_in_tile, px_in_tile};

    // ------------------------------------------------------------------
    // ROMs
    // ------------------------------------------------------------------
    maze_rom u_maze_rom (
        .clk       (clk),
        .addr      (maze_addr),
        .cell_type (maze_cell)
    );

    sprite_rom #(.INIT_FILE("evader.hex")) u_evader_rom (
        .clk(clk), .addr(sprite_addr), .pixel_color(evader_pixel));
    sprite_rom #(.INIT_FILE("ghost0.hex")) u_ghost0_rom (
        .clk(clk), .addr(sprite_addr), .pixel_color(ghost0_pixel));
    sprite_rom #(.INIT_FILE("ghost1.hex")) u_ghost1_rom (
        .clk(clk), .addr(sprite_addr), .pixel_color(ghost1_pixel));

    // ------------------------------------------------------------------
    // Compositor + palette
    // ------------------------------------------------------------------
    compositor u_compositor (
        .VGA_BLANK_n     (VGA_BLANK_n),
        .in_maze         (in_maze),
        .tile_col        (tile_col),
        .tile_row        (tile_row),
        .maze_cell       (maze_cell),
        .evader_pixel    (evader_pixel),
        .ghost0_pixel    (ghost0_pixel),
        .ghost1_pixel    (ghost1_pixel),
        .evader_col      (evader_col),
        .evader_row      (evader_row),
        .ghost0_col      (ghost0_col),
        .ghost0_row      (ghost0_row),
        .ghost1_col      (ghost1_col),
        .ghost1_row      (ghost1_row),
        .tile_activated  (tile_activated),
        .gate_open       (game_status_q[0]),
        .evader_win      (game_status_q[1]),
        .ghost_win       (game_status_q[2]),
        .gate_col        (GATE_COL),
        .gate_row        (GATE_ROW),
        .VGA_R           (VGA_R),
        .VGA_G           (VGA_G),
        .VGA_B           (VGA_B)
    );

endmodule


// ------------------------------------------------------------------
```

```
// vga_counters - 640x480 @ 60 Hz timing for a 50 MHz clock.  Lifted
// from the lab3 starter and trimmed.
// ---------------------------------------------------------------------
module vga_counters(
    input  logic        clk50, reset,
    output logic [10:0] hcount,
    output logic [9:0]  vcount,
    output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n,
    output logic        vsync_pulse
);

    parameter HACTIVE      = 11'd1280,
              HFRONT_PORCH = 11'd32,
              HSYNC        = 11'd192,
              HBACK_PORCH  = 11'd96,
              HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH;

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

    logic endOfLine, endOfField;

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

    assign endOfLine = hcount == HTOTAL - 1;

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

    assign endOfField = vcount == VTOTAL - 1;

    // Single-cycle pulse at start of frame (used for vsync_cdc)
    assign vsync_pulse = endOfLine && endOfField;

    assign VGA_HS = !((hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
    assign VGA_VS = !(vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
    assign VGA_SYNC_n  = 1'b0;
    assign VGA_BLANK_n = !(hcount[10] & (hcount[9] | hcount[8])) &
                         !(vcount[9] | (vcount[8:5] == 4'b1111));
    assign VGA_CLK = hcount[0];
endmodule


// ---------------------------------------------------------------------
// vsync_cdc - 2-FF synchroniser + rising-edge detect.  The source pulse
// comes from the pixel-clock VGA counter; we want a single-cycle strobe
// in the system clk domain.
// ---------------------------------------------------------------------
module vsync_cdc(
    input  logic clk,
```

```
    input  logic reset,
    input  logic vsync_pulse,
    output logic vsync_tick
);
    logic meta_q, sync_q, prev_q;
    always_ff @(posedge clk) begin
        if (reset) begin
            meta_q <= 1'b0;
            sync_q <= 1'b0;
            prev_q <= 1'b0;
        end else begin
            meta_q <= vsync_pulse;
            sync_q <= meta_q;
            prev_q <= sync_q;
        end
    end
    assign vsync_tick = sync_q & ~prev_q;
endmodule


// ---------------------------------------------------------------------
// tile_timer - 14-bit saturating counter, increments while the Evader is
// on the parameterised (TILE_COL, TILE_ROW) tile and we receive a
// vsync_tick.  Threshold is 180 frames (~3 s at 60 Hz); we shortened
// this from the original 600 frames so the demo is paced reasonably.
// ---------------------------------------------------------------------
module tile_timer #(
    parameter logic [4:0] TILE_COL = 5'd0,
    parameter logic [4:0] TILE_ROW = 5'd0
) (
    input  logic        clk,
    input  logic        reset,
    input  logic        vsync_tick,
    input  logic [4:0]  evader_col,
    input  logic [4:0]  evader_row,
    output logic [13:0] count,
    output logic        activated
);
    logic on_tile;
    assign on_tile   = (evader_col == TILE_COL) && (evader_row == TILE_ROW);
    assign activated = (count >= 14'd180);

    always_ff @(posedge clk) begin
        if (reset)
            count <= 14'd0;
        else if (vsync_tick && on_tile && !activated)
            count <= count + 14'd1;
    end
endmodule


// ---------------------------------------------------------------------
// maze_rom - 600 x 3-bit synchronous ROM.  Cell encoding:
//   3'd0 wall  3'd1 path  3'd2 special  3'd3 gate
// Initialised from maze.hex (generated by gen_mif.py).
// ---------------------------------------------------------------------
module maze_rom(
    input  logic       clk,
    input  logic [9:0] addr,
```

```
    output logic [2:0] cell_type
);
    (* ramstyle = "M10K" *) logic [2:0] mem [0:599];
    initial $readmemh("maze.hex", mem);

    always_ff @(posedge clk)
        cell_type <= mem[addr];
endmodule


// ------------------------------------------------------------------------
// sprite_rom - 256 x 3-bit synchronous ROM (16x16 sprite).  Index 0 is
// treated as transparent by the compositor.
// ------------------------------------------------------------------------
module sprite_rom #(parameter INIT_FILE = "evader.hex") (
    input  logic       clk,
    input  logic [7:0] addr,
    output logic [2:0] pixel_color
);
    (* ramstyle = "M10K" *) logic [2:0] mem [0:255];
    initial $readmemh(INIT_FILE, mem);

    always_ff @(posedge clk)
        pixel_color <= mem[addr];
endmodule


// ------------------------------------------------------------------------
// compositor - picks the final RGB value for the current pixel.
//
// Pixel priority:
//   1. !VGA_BLANK_n            -> black
//   2. game_result asserted    -> full-screen overlay (green = evader win,
//                                                      red   = ghost  win)
//   3. !in_maze                -> black border around the playfield
//   4. Sprite layer (evader > ghost0 > ghost1; nonzero index wins)
//   5. Tile layer: cell_type -> palette colour, modulated by
//                  tile_activated[] and gate_open.
// ------------------------------------------------------------------------
module compositor(
    input  logic        VGA_BLANK_n,
    input  logic        in_maze,
    input  logic [4:0]  tile_col, tile_row,
    input  logic [2:0]  maze_cell,
    input  logic [2:0]  evader_pixel,
    input  logic [2:0]  ghost0_pixel,
    input  logic [2:0]  ghost1_pixel,
    input  logic [4:0]  evader_col, evader_row,
    input  logic [4:0]  ghost0_col, ghost0_row,
    input  logic [4:0]  ghost1_col, ghost1_row,
    input  logic [5:0]  tile_activated,
    input  logic        gate_open,
    input  logic        evader_win,
    input  logic        ghost_win,
    input  logic [4:0]  gate_col, gate_row,
    output logic [7:0]  VGA_R, VGA_G, VGA_B
);
    // ------ identify which tile (if any) we are currently on ---------
    logic on_evader, on_ghost0, on_ghost1;
```

```
    assign on_evader = (tile_col == evader_col) && (tile_row == evader_row);
    assign on_ghost0 = (tile_col == ghost0_col) && (tile_row == ghost0_row);
    assign on_ghost1 = (tile_col == ghost1_col) && (tile_row == ghost1_row);

    // ------ identify if current tile is one of the 6 special tiles --
    logic        is_special_tile;
    logic [2:0]  special_idx;
    always_comb begin
        is_special_tile = 1'b0;
        special_idx     = 3'd0;
        case ({tile_col, tile_row})
            {5'd3 , 5'd2 }: begin is_special_tile = 1'b1; special_idx = 3'd0; end
            {5'd26, 5'd2 }: begin is_special_tile = 1'b1; special_idx = 3'd1; end
            {5'd3 , 5'd10}: begin is_special_tile = 1'b1; special_idx = 3'd2; end
            {5'd26, 5'd10}: begin is_special_tile = 1'b1; special_idx = 3'd3; end
            {5'd14, 5'd17}: begin is_special_tile = 1'b1; special_idx = 3'd4; end
            {5'd17, 5'd17}: begin is_special_tile = 1'b1; special_idx = 3'd5; end
            default: ;
        endcase
    end
    logic this_tile_active;
    assign this_tile_active = is_special_tile && tile_activated[special_idx];
    logic is_gate_tile;
    assign is_gate_tile = (tile_col == gate_col) && (tile_row == gate_row);

    // ------ tile background colour (from maze_cell + activation) ----
    logic [23:0] tile_rgb;
    always_comb begin
        unique case (maze_cell)
            3'd0:    tile_rgb = 24'h202050;                  // WALL   : dark blue
            3'd1:    tile_rgb = 24'h000000;                  // PATH   : black
            3'd2:    tile_rgb = this_tile_active ? 24'h60E060   // active green
                                                 : 24'h605030;  // dim yellow
            3'd3:    tile_rgb = gate_open       ? 24'h60E060   // gate open
                                                 : 24'h606060;  // grey
            default: tile_rgb = 24'h000000;
        endcase
    end

    // ------ sprite colour palette (3-bit index)  --------------------
    // 0 = transparent (handled by compositor mux)
    // 1..7 = colours.  Each sprite ROM has its own meaning per index but
    // we share a single palette here.  Evader uses warm colours; ghost0
    // bluish; ghost1 reddish.  The colour wheels are encoded in the .hex
    // files; we just need a way to turn a 3-bit value into RGB for each
    // entity.  To keep the compositor simple we use the same palette and
    // let the .hex contents do the differentiation.
    function automatic [23:0] palette(input logic [2:0] idx);
        case (idx)
            3'd0: palette = 24'h000000;  // transparent (never selected)
            3'd1: palette = 24'hFFFFFF;  // white
            3'd2: palette = 24'hFFD030;  // gold (evader body)
            3'd3: palette = 24'hC04020;  // red-brown (evader outline)
            3'd4: palette = 24'h4080FF;  // blue   (ghost0 body)
            3'd5: palette = 24'h2040A0;  // dk blue (ghost0 outline)
            3'd6: palette = 24'hFF6060;  // red    (ghost1 body)
            3'd7: palette = 24'hA02020;  // dk red (ghost1 outline)
            default: palette = 24'h000000;
        endcase
```

```
529     endfunction
530
531     // ------ pick which entity (if any) covers this pixel -----------
532     logic [2:0] sprite_pix;
533     always_comb begin
534         sprite_pix = 3'd0;
535         if (on_evader && evader_pixel != 3'd0)
536             sprite_pix = evader_pixel;
537         else if (on_ghost0 && ghost0_pixel != 3'd0)
538             sprite_pix = ghost0_pixel;
539         else if (on_ghost1 && ghost1_pixel != 3'd0)
540             sprite_pix = ghost1_pixel;
541     end
542
543     logic [23:0] pixel_rgb;
544     always_comb begin
545         if (!VGA_BLANK_n) begin
546             pixel_rgb = 24'h000000;
547         end else if (evader_win) begin
548             pixel_rgb = 24'h00B000;       // green overlay
549         end else if (ghost_win) begin
550             pixel_rgb = 24'hB00000;       // red overlay
551         end else if (!in_maze) begin
552             pixel_rgb = 24'h000000;       // black border
553         end else if (sprite_pix != 3'd0) begin
554             pixel_rgb = palette(sprite_pix);
555         end else begin
556             pixel_rgb = tile_rgb;
557         end
558     end
559
560     assign {VGA_R, VGA_G, VGA_B} = pixel_rgb;
561 endmodule
```

A.3 Hardware: soc_system_top.sv

```
// soc_system_top - DE1-SoC board-level wrapper.
// Adapted from the lab3 starter.  The Qsys-generated `soc_system`
// instance contains the HPS and our `game_peripheral`; only the VGA
// conduit gets exported, so it is wired straight to the on-board DAC
// pins.  All other (unused) peripherals are tied off the same way the
// lab3 template does it to avoid Quartus "no driver" warnings.

module soc_system_top(
    ///////// ADC /////////
    inout          ADC_CS_N,
    output         ADC_DIN,
    input          ADC_DOUT,
    output         ADC_SCLK,

    ///////// AUD /////////
    input          AUD_ADCDAT,
    inout          AUD_ADCLRCK,
    inout          AUD_BCLK,
    output         AUD_DACDAT,
    inout          AUD_DACLRCK,
    output         AUD_XCK,

    ///////// CLOCK2/3/4 /////////
    input          CLOCK2_50,
    input          CLOCK3_50,
    input          CLOCK4_50,

    ///////// CLOCK /////////
    input          CLOCK_50,

    ///////// DRAM /////////
    output [12:0] DRAM_ADDR,
    output [1:0]  DRAM_BA,
    output         DRAM_CAS_N,
    output         DRAM_CKE,
    output         DRAM_CLK,
    output         DRAM_CS_N,
    inout  [15:0] DRAM_DQ,
    output         DRAM_LDQM,
    output         DRAM_RAS_N,
    output         DRAM_UDQM,
    output         DRAM_WE_N,

    ///////// FAN /////////
    output         FAN_CTRL,

    ///////// FPGA /////////
    output         FPGA_I2C_SCLK,
    inout          FPGA_I2C_SDAT,

    ///////// GPIO /////////
    inout  [35:0] GPIO_0,
    inout  [35:0] GPIO_1,

    ///////// HEX /////////
    output [6:0]  HEX0,
```

```
57      output  [6:0]   HEX1,
58      output  [6:0]   HEX2,
59      output  [6:0]   HEX3,
60      output  [6:0]   HEX4,
61      output  [6:0]   HEX5,
62
63      ///////// HPS /////////
64      inout           HPS_CONV_USB_N,
65      output  [14:0]  HPS_DDR3_ADDR,
66      output  [2:0]   HPS_DDR3_BA,
67      output          HPS_DDR3_CAS_N,
68      output          HPS_DDR3_CKE,
69      output          HPS_DDR3_CK_N,
70      output          HPS_DDR3_CK_P,
71      output          HPS_DDR3_CS_N,
72      output  [3:0]   HPS_DDR3_DM,
73      inout   [31:0]  HPS_DDR3_DQ,
74      inout   [3:0]   HPS_DDR3_DQS_N,
75      inout   [3:0]   HPS_DDR3_DQS_P,
76      output          HPS_DDR3_ODT,
77      output          HPS_DDR3_RAS_N,
78      output          HPS_DDR3_RESET_N,
79      input           HPS_DDR3_RZQ,
80      output          HPS_DDR3_WE_N,
81      output          HPS_ENET_GTX_CLK,
82      inout           HPS_ENET_INT_N,
83      output          HPS_ENET_MDC,
84      inout           HPS_ENET_MDIO,
85      input           HPS_ENET_RX_CLK,
86      input   [3:0]   HPS_ENET_RX_DATA,
87      input           HPS_ENET_RX_DV,
88      output  [3:0]   HPS_ENET_TX_DATA,
89      output          HPS_ENET_TX_EN,
90      inout           HPS_GSENSOR_INT,
91      inout           HPS_I2C1_SCLK,
92      inout           HPS_I2C1_SDAT,
93      inout           HPS_I2C2_SCLK,
94      inout           HPS_I2C2_SDAT,
95      inout           HPS_I2C_CONTROL,
96      inout           HPS_KEY,
97      inout           HPS_LED,
98      inout           HPS_LTC_GPIO,
99      output          HPS_SD_CLK,
100     inout           HPS_SD_CMD,
101     inout   [3:0]   HPS_SD_DATA,
102     output          HPS_SPIM_CLK,
103     input           HPS_SPIM_MISO,
104     output          HPS_SPIM_MOSI,
105     inout           HPS_SPIM_SS,
106     input           HPS_UART_RX,
107     output          HPS_UART_TX,
108     input           HPS_USB_CLKOUT,
109     inout   [7:0]   HPS_USB_DATA,
110     input           HPS_USB_DIR,
111     input           HPS_USB_NXT,
112     output          HPS_USB_STP,
113
114     ///////// IRDA /////////
115     input           IRDA_RXD,
```

```
    output          IRDA_TXD,

    ///////// KEY /////////
    input   [3:0]   KEY,

    ///////// LEDR /////////
    output  [9:0]   LEDR,

    ///////// PS2 /////////
    inout           PS2_CLK,
    inout           PS2_CLK2,
    inout           PS2_DAT,
    inout           PS2_DAT2,

    ///////// SW /////////
    input   [9:0]   SW,

    ///////// TD /////////
    input           TD_CLK27,
    input   [7:0]   TD_DATA,
    input           TD_HS,
    output          TD_RESET_N,
    input           TD_VS,

    ///////// VGA /////////
    output  [7:0]   VGA_B,
    output          VGA_BLANK_N,
    output          VGA_CLK,
    output  [7:0]   VGA_G,
    output          VGA_HS,
    output  [7:0]   VGA_R,
    output          VGA_SYNC_N,
    output          VGA_VS
);

   soc_system soc_system0(
     .clk_clk                       ( CLOCK_50 ),
     .reset_reset_n                 ( 1'b1 ),

     .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
     .hps_ddr3_mem_dm               ( HPS_DDR3_DM ),
     .hps_ddr3_oct_rzqin            ( HPS_DDR3_RZQ ),

     .hps_hps_io_emac1_inst_TX_CLK  ( HPS_ENET_GTX_CLK ),
     .hps_hps_io_emac1_inst_TXD0    ( HPS_ENET_TX_DATA[0] ),
     .hps_hps_io_emac1_inst_TXD1    ( HPS_ENET_TX_DATA[1] ),
```

```
175       .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
176       .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
177       .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
178       .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
179       .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
180       .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
181       .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
182       .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
183       .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
184       .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
185       .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),
186
187       .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD     ),
188       .hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0] ),
189       .hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1] ),
190       .hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK     ),
191       .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2] ),
192       .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3] ),
193
194       .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0] ),
195       .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1] ),
196       .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2] ),
197       .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3] ),
198       .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4] ),
199       .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5] ),
200       .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6] ),
201       .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7] ),
202       .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT  ),
203       .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP     ),
204       .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR     ),
205       .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT     ),
206
207       .hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
208       .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
209       .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
210       .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),
211
212       .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX ),
213       .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX ),
214
215       .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT ),
216       .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK ),
217       .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT ),
218       .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK ),
219
220       .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
221       .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
222       .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),
223       .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
224       .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
225       .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
226       .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),
227
228       // VGA conduit from custom game_peripheral
229       .vga_r        (VGA_R),
230       .vga_g        (VGA_G),
231       .vga_b        (VGA_B),
232       .vga_clk      (VGA_CLK),
233       .vga_hs       (VGA_HS),
```

```
      .vga_vs       (VGA_VS),
      .vga_blank_n  (VGA_BLANK_N),
      .vga_sync_n   (VGA_SYNC_N)
  );

   // Quiet "no driver" warnings for the unused board peripherals.
   assign ADC_CS_N    = SW[1] ? SW[0] : 1'bZ;
   assign ADC_DIN     = SW[0];
   assign ADC_SCLK    = SW[0];

   assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
   assign AUD_BCLK    = SW[1] ? SW[0] : 1'bZ;
   assign AUD_DACDAT  = SW[0];
   assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
   assign AUD_XCK     = SW[0];

   assign DRAM_ADDR   = { 13{ SW[0] } };
   assign DRAM_BA     = { 2{ SW[0] } };
   assign DRAM_DQ     = SW[1] ? { 16{ SW[0] } } : { 16{ 1'bZ } };
   assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
           DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

   assign FAN_CTRL       = SW[0];
   assign FPGA_I2C_SCLK  = SW[0];
   assign FPGA_I2C_SDAT  = SW[1] ? SW[0] : 1'bZ;

   assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };
   assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };

   assign HEX0 = { 7{ SW[1] } };
   assign HEX1 = { 7{ SW[2] } };
   assign HEX2 = { 7{ SW[3] } };
   assign HEX3 = { 7{ SW[4] } };
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

   assign IRDA_TXD = SW[0];
   assign LEDR     = { 10{SW[7]} };

   assign PS2_CLK  = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT  = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

   assign TD_RESET_N = SW[0];

endmodule
```

A.4 Hardware: game_peripheral_hw.tcl

```
# TCL file describing the game_peripheral component to Platform Designer.
# Mirrors lab3/vga_ball_hw.tcl but with a 32-bit Avalon-MM slave and the
# four-bit register-file address bus.

package require -exact qsys 16.1

# ------------------------------------------------------------------------
# module game_peripheral
# ------------------------------------------------------------------------
set_module_property DESCRIPTION ""
set_module_property NAME game_peripheral
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 "Maze Chase Game Peripheral"
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

set_module_assignment embeddedsw.dts.vendor "csee4840"
set_module_assignment embeddedsw.dts.name   "game_peripheral"
set_module_assignment embeddedsw.dts.group  "game"

# ------------------------------------------------------------------------
# file sets
# ------------------------------------------------------------------------
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL game_peripheral
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file game_peripheral.sv SYSTEM_VERILOG PATH game_peripheral.sv
    TOP_LEVEL_FILE
add_fileset_file maze.hex   OTHER PATH maze.hex
add_fileset_file evader.hex OTHER PATH evader.hex
add_fileset_file ghost0.hex OTHER PATH ghost0.hex
add_fileset_file ghost1.hex OTHER PATH ghost1.hex

# ------------------------------------------------------------------------
# clock
# ------------------------------------------------------------------------
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
add_interface_port clock clk clk Input 1

# ------------------------------------------------------------------------
# reset
# ------------------------------------------------------------------------
add_interface reset reset end
set_interface_property reset associatedClock     clock
set_interface_property reset synchronousEdges    DEASSERT
set_interface_property reset ENABLED             true
add_interface_port reset reset reset Input 1
```

```
56
57 # ------------------------------------------------------------------------
58 # avalon_slave_0  (32-bit data, 16 words = 64 bytes)
59 # ------------------------------------------------------------------------
60 add_interface avalon_slave_0 avalon end
61 set_interface_property avalon_slave_0 addressUnits              WORDS
62 set_interface_property avalon_slave_0 associatedClock          clock
63 set_interface_property avalon_slave_0 associatedReset          reset
64 set_interface_property avalon_slave_0 bitsPerSymbol            8
65 set_interface_property avalon_slave_0 burstOnBurstBoundariesOnly false
66 set_interface_property avalon_slave_0 burstcountUnits          WORDS
67 set_interface_property avalon_slave_0 explicitAddressSpan      0
68 set_interface_property avalon_slave_0 holdTime                 0
69 set_interface_property avalon_slave_0 linewrapBursts           false
70 set_interface_property avalon_slave_0 maximumPendingReadTransactions 0
71 set_interface_property avalon_slave_0 maximumPendingWriteTransactions 0
72 set_interface_property avalon_slave_0 readLatency              0
73 set_interface_property avalon_slave_0 readWaitTime             1
74 set_interface_property avalon_slave_0 setupTime                0
75 set_interface_property avalon_slave_0 timingUnits             Cycles
76 set_interface_property avalon_slave_0 writeWaitTime           0
77 set_interface_property avalon_slave_0 ENABLED                  true
78
79 add_interface_port avalon_slave_0 writedata   writedata   Input  32
80 add_interface_port avalon_slave_0 readdata    readdata    Output 32
81 add_interface_port avalon_slave_0 write       write       Input  1
82 add_interface_port avalon_slave_0 read        read        Input  1
83 add_interface_port avalon_slave_0 chipselect  chipselect  Input  1
84 add_interface_port avalon_slave_0 address     address     Input  4
85
86 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isFlash
                0
87 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isMemoryDevice
        0
88 set_interface_assignment avalon_slave_0
    embeddedsw.configuration.isNonVolatileStorage 0
89 set_interface_assignment avalon_slave_0
    embeddedsw.configuration.isPrintableDevice    0
90
91 # ------------------------------------------------------------------------
92 # vga conduit
93 # ------------------------------------------------------------------------
94 add_interface vga conduit end
95 set_interface_property vga associatedClock clock
96 set_interface_property vga associatedReset ""
97 set_interface_property vga ENABLED         true
98
99 add_interface_port vga VGA_R       r       Output 8
100 add_interface_port vga VGA_G       g       Output 8
101 add_interface_port vga VGA_B       b       Output 8
102 add_interface_port vga VGA_CLK     clk     Output 1
103 add_interface_port vga VGA_HS      hs      Output 1
104 add_interface_port vga VGA_VS      vs      Output 1
105 add_interface_port vga VGA_BLANK_n blank_n Output 1
106 add_interface_port vga VGA_SYNC_n  sync_n  Output 1
```

A.5 Hardware: gen_mif.py

```
#!/usr/bin/env python3
"""gen_mif.py - build the ROM init files used by the Maze Chase Game.

Outputs (written next to this script):
    maze.mif / maze.hex          600 x 3-bit  Maze ROM
    evader.mif / evader.hex      256 x 3-bit  Evader sprite
    ghost0.mif / ghost0.hex      256 x 3-bit  Ghost 0 sprite
    ghost1.mif / ghost1.hex      256 x 3-bit  Ghost 1 sprite
    maze_data.h                  C header consumed by the HPS app

The .mif files are picked up by Quartus altsyncram instantiation; the
.hex files exist so that $readmemh in the SystemVerilog ROMs works
identically inside simulators that do not parse .mif.

Cell encoding (matches game_peripheral.sv):
    0 wall   1 path   2 special   3 gate

The maze layout is a hand-authored 30x20 grid that satisfies the
constraints from the design document:
    - 6 special tiles at (3,2) (26,2) (3,10) (26,10) (14,17) (17,17)
    - Gate tile at (15,1)
    - Outer border wall
    - All special tiles reachable from the Evader's spawn at (15,10)
"""

import os
from collections import deque

# ---------------------------------------------------------------------------
# Maze layout (30 cols x 20 rows) - programmatically built.
#  * Outer border = wall
#  * Irregular interior wall pattern (more random feel than the old
#    uniform pillar grid) - hand-authored but verified at build time.
#  * Special tiles forced at design-spec coordinates
#  * Gate forced at design-spec coordinate
# ---------------------------------------------------------------------------
COLS = 30
ROWS = 20

# Inner-wall mask: 'W' means "place a wall here" before specials/gate are
# stamped on top.  '.' (or anything that is not 'W') means leave the cell
# as path.  Row 0, row 1 and row ROWS-1 are overwritten by build_maze()
# (outer border + gate row), so the contents of those rows in the mask
# are ignored.
#
# Columns: 0         1         2
#          012345678901234567890123456789
WALL_MASK = [
    "..............................",  # 0
    "..............................",  # 1
    "..............................",  # 2  open corridor (specials at 3, 26)
    "..W.WW.W.W.W.WW.W.W.WW.W.W.WW.",  # 3
    "..W..W.W.W..W.W.W..W.W.W..W.",  # 4
    "..WW.W.W.WWW..W.W.WW.W.W.WW.W.",  # 5
    ".....W.W......W.W....W.W......",  # 6
    "..WWW.W.WWWWW.W.WWW.W.W.W.WWW.",  # 7
```

```
    "..W...W.....W.W...W.W.W.W...W.",  # 8
    "..W.WWW.WWW.W.WWW.W.W.W.WWW.W.",  # 9
    "............................",  # 10 open corridor (specials at 3, 26)
    "..W.WWW.W.WW.W.WW.W.WWW.W.WW.W",  # 11
    "..W...W.W..W.W..W.W...W.W..W.W",  # 12
    "..WWW.W.WW.W.WW.W.WWW.W.WW.W.W",  # 13
    "....W.W..W.W.W..W.....W..W....",  # 14
    "..W.W.WWW.W.W.WWWW.WWWW.W.WWW.",  # 15
    "..W.W...W.W.W....W....W.W...W.",  # 16
    "..W.WWW.W.W.WWW.WWW.W.W.W.W.W.",  # 17
    "............................",  # 18 open corridor (specials at 14, 17)
    "............................",  # 19
]


def build_maze():
    """Returns the 20x30 maze grid prior to placing specials/gate."""
    maze = [[1] * COLS for _ in range(ROWS)]
    # Outer wall
    for c in range(COLS):
        maze[0][c] = 0
        maze[ROWS - 1][c] = 0
    for r in range(ROWS):
        maze[r][0] = 0
        maze[r][COLS - 1] = 0
    # Inner walls from the mask
    for r, row in enumerate(WALL_MASK):
        for c, ch in enumerate(row[:COLS].ljust(COLS, ".")):
            if ch == "W":
                maze[r][c] = 0
    # Row 1 is solid wall *except* at the gate column (force later)
    for c in range(COLS):
        maze[1][c] = 0
    return maze


def force_special(maze, specials):
    for col, row in specials:
        maze[row][col] = 2


def force_gate(maze, gate):
    col, row = gate
    maze[row][col] = 3


SPECIALS = [(3, 2), (26, 2), (3, 10), (26, 10), (14, 17), (17, 17)]
GATE = (15, 1)

# Cells that MUST be reachable from the evader's spawn through non-wall
# cells.  The gate itself is reached from (15, 2); the gate cell is path
# only after gate_open, so we instead require (15, 2) to be a path.
EVADER_START = (15, 10)
GHOST_STARTS = [(1, 2), (28, 2)]
GATE_NEIGHBOR = (15, 2)
REACHABILITY_TARGETS = SPECIALS + GHOST_STARTS + [GATE_NEIGHBOR]


def bfs_reachable(maze, start):
```

```
    """Return the set of (col, row) reachable from start via non-wall cells."""
    visited = {start}
    q = deque([start])
    while q:
        c, r = q.popleft()
        for dc, dr in ((0, -1), (0, 1), (-1, 0), (1, 0)):
            nc, nr = c + dc, r + dr
            if 0 <= nc < COLS and 0 <= nr < ROWS and (nc, nr) not in visited \
                    and maze[nr][nc] != 0:
                visited.add((nc, nr))
                q.append((nc, nr))
    return visited


def carve_to(maze, start, target):
    """If target is unreachable, knock walls along the straight-line path
    between start and target so that target becomes reachable.  Used as a
    safety net so the maze is always playable even if the WALL_MASK above
    accidentally seals off a critical cell."""
    sc, sr = start
    tc, tr = target
    # Walk col first, then row.  Any wall we hit on the way becomes path.
    c = sc
    while c != tc:
        c += 1 if tc > c else -1
        if maze[sr][c] == 0:
            maze[sr][c] = 1
    r = sr
    while r != tr:
        r += 1 if tr > r else -1
        if maze[r][tc] == 0:
            maze[r][tc] = 1


def ensure_connectivity(maze):
    """BFS from EVADER_START; carve corridors to any critical cell that
    isn't reachable.  Repeats until everything is connected."""
    for _ in range(8):
        reachable = bfs_reachable(maze, EVADER_START)
        missing = [p for p in REACHABILITY_TARGETS if p not in reachable]
        if not missing:
            return
        for tgt in missing:
            carve_to(maze, EVADER_START, tgt)
    raise RuntimeError("Could not reach all critical cells from evader start")


def parse_maze():
    maze = build_maze()
    # Stamp specials and gate FIRST so the connectivity check sees them as
    # path-equivalent cells (anything != 0 is treated as walkable).
    force_special(maze, SPECIALS)
    force_gate(maze, GATE)
    ensure_connectivity(maze)
    return maze


# ---------------------------------------------------------------------------
# 16x16 sprites
```

```
#   Each is a list of 16 strings, 16 chars each.
#   Character meaning (matches palette() in game_peripheral.sv):
#     '.' = 0 transparent
#     '1' = white
#     '2' = evader body  (gold)
#     '3' = evader outline (red-brown)
#     '4' = ghost0 body  (blue)
#     '5' = ghost0 outline (dark blue)
#     '6' = ghost1 body  (red)
#     '7' = ghost1 outline (dark red)
# ------------------------------------------------------------------------
EVADER = [
    "................",
    ".....333333.....",
    "....32222223....",
    "...3222222223...",
    "..322221122223..",
    "..322211112223..",
    "..322111111223..",
    "..321111111123..",
    "..321111111123..",
    "..322111111223..",
    "..322211112223..",
    "..322221122223..",
    "...3222222223...",
    "....32222223....",
    ".....333333.....",
    "................",
]

GHOST0 = [
    "................",
    "....555555......",
    "...54444445.....",
    "..5444444445....",
    "..5441441445....",
    "..5411111145....",
    "..5411111145....",
    "..5444444445....",
    "..5444444445....",
    "..5444444445....",
    "..5444444445....",
    "..5444444445....",
    "..5454545455....",
    "..5454545455....",
    "...545454545....",
    "................",
]

GHOST1 = [
    "................",
    "....777777......",
    "...76666667.....",
    "..7666666667....",
    "..7661661667....",
    "..7611111167....",
    "..7611111167....",
    "..7666666667....",
    "..7666666667....",
```

```
    "..7666666667....",
    "..7666666667....",
    "..7666666667....",
    "..7676767677....",
    "..7676767677....",
    "...767676767....",
    "................",
]


def parse_sprite(pixels):
    out = []
    assert len(pixels) == 16
    for row in pixels:
        row = row[:16].ljust(16, ".")
        for ch in row:
            if ch == ".":
                out.append(0)
            else:
                out.append(int(ch, 16))
    assert len(out) == 256
    return out


# ---------------------------------------------------------------------------
# Writers
# ---------------------------------------------------------------------------
def write_mif(path, data, width, depth):
    with open(path, "w") as f:
        f.write(f"WIDTH={width};\nDEPTH={depth};\n")
        f.write("ADDRESS_RADIX=UNS;\nDATA_RADIX=UNS;\nCONTENT BEGIN\n")
        for i, v in enumerate(data):
            f.write(f"  {i}\t:   {v};\n")
        f.write("END;\n")


def write_hex(path, data):
    # one nibble per line is enough for 3-bit values; readmemh tolerates this
    with open(path, "w") as f:
        for v in data:
            f.write(f"{v:x}\n")


def write_c_header(path, maze):
    with open(path, "w") as f:
        f.write("/* Auto-generated by gen_mif.py - do not edit */\n")
        f.write("#ifndef MAZE_DATA_H\n#define MAZE_DATA_H\n\n")
        f.write("#define MAZE_COLS 30\n")
        f.write("#define MAZE_ROWS 20\n\n")
        f.write("static const unsigned char maze_data[MAZE_ROWS][MAZE_COLS] =
            {\n")
        for row in maze:
            f.write("    { " + ", ".join(str(v) for v in row) + " },\n")
        f.write("};\n\n#endif\n")


def main():
    here = os.path.dirname(os.path.abspath(__file__))
    maze = parse_maze()
```

```
    flat = [v for row in maze for v in row]
    write_mif(os.path.join(here, "maze.mif"), flat, 3, 600)
    write_hex(os.path.join(here, "maze.hex"), flat)

    for name, sprite in [("evader", EVADER),
                         ("ghost0", GHOST0),
                         ("ghost1", GHOST1)]:
        data = parse_sprite(sprite)
        write_mif(os.path.join(here, f"{name}.mif"), data, 3, 256)
        write_hex(os.path.join(here, f"{name}.hex"), data)

    # Maze data header for the HPS C application
    sw_dir = os.path.abspath(os.path.join(here, "..", "final_project-sw"))
    if os.path.isdir(sw_dir):
        write_c_header(os.path.join(sw_dir, "maze_data.h"), maze)
    write_c_header(os.path.join(here, "maze_data.h"), maze)

    print("Generated:")
    for fn in ("maze.mif", "maze.hex", "evader.mif", "evader.hex",
               "ghost0.mif", "ghost0.hex", "ghost1.mif", "ghost1.hex",
               "maze_data.h"):
        print(" ", fn)


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

A.6 Hardware: Makefile

```
# ============================================================================
# Hardware Makefile - Maze Chase Game (DE1-SoC)
#
# Mirrors lab3/lab3-hw/Makefile but with the new peripheral / source
# files.  Run `make project` first to bootstrap Quartus, then `make
# qsys`, `make quartus`, `make rbf`, etc.
# ============================================================================
SYSTEM   = soc_system

TCL      = $(SYSTEM).tcl
QSYS     = $(SYSTEM).qsys
SOPCINFO = $(SYSTEM).sopcinfo
QIP      = $(SYSTEM)/synthesis/$(SYSTEM).qip
HPS_PIN_TCL = $(SYSTEM)/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
HPS_PIN_MAP = hps_sdram_p0_all_pins.txt
QPF      = $(SYSTEM).qpf
QSF      = $(SYSTEM).qsf
SDC      = $(SYSTEM).sdc
SRF      = $(SYSTEM).srf

BOARD_INFO = $(SYSTEM)_board_info.xml
DTS = $(SYSTEM).dts
DTB = $(SYSTEM).dtb

SOF = output_files/$(SYSTEM).sof
RBF = output_files/$(SYSTEM).rbf

# ROM init files - regenerated by gen_mif.py.
ROM_FILES = maze.mif maze.hex \
            evader.mif evader.hex \
            ghost0.mif ghost0.hex \
            ghost1.mif ghost1.hex \
            maze_data.h

TARFILES = Makefile \
    $(TCL) \
    $(QSYS) \
    $(SYSTEM)_top.sv \
    $(BOARD_INFO) \
    $(SRF) \
    game_peripheral.sv \
    game_peripheral_hw.tcl \
    gen_mif.py \
    $(ROM_FILES)

TARFILE = final_project-hw.tar.gz

# ----------------------------------------------------------------------------
# ROM data generation
# ----------------------------------------------------------------------------
.PHONY: roms
roms: $(ROM_FILES)

$(ROM_FILES): gen_mif.py
    python3 gen_mif.py

```

```
# ---------------------------------------------------------------------
# project
# ---------------------------------------------------------------------
.PHONY: project
project: $(QPF) $(QSF) $(SDC)

$(QPF) $(QSF) $(SDC) : $(TCL) $(HPS_PIN_TCL)
    quartus_sh -t $(TCL)
    quartus_map $(SYSTEM)
    quartus_sta -t $(HPS_PIN_TCL) $(SYSTEM)

# ---------------------------------------------------------------------
# qsys
# ---------------------------------------------------------------------
.PHONY: qsys
qsys: $(SOPCINFO)

$(SOPCINFO) $(QIP) $(HPS_PIN_TCL) $(SYSTEM)/ : $(QSYS) $(ROM_FILES)
    rm -rf $(SOPCINFO) $(SYSTEM)/
    qsys-generate $(QSYS) --synthesis=VERILOG

# ---------------------------------------------------------------------
# quartus
# ---------------------------------------------------------------------
.PHONY: quartus
quartus: $(SOF)

$(SOF) $(HPS_PIN_MAP) : $(QIP) $(QPF) $(QSF) $(HPS_PIN_TCL)
    quartus_sh --flow compile $(QPF)

# ---------------------------------------------------------------------
# rbf - converts .sof to .rbf for SD-card-based programming
# ---------------------------------------------------------------------
.PHONY: rbf
rbf: $(RBF)

$(RBF): $(SOF)
    quartus_cpf -c $(SOF) $(RBF)

# ---------------------------------------------------------------------
# dtb
# ---------------------------------------------------------------------
.PHONY: dtb
dtb: $(DTB)

$(DTB): $(DTS)
    @which dtc || (echo "dtc not found.  Did you run embedded_command_shell.sh?";
        exit 1)
    dtc -I dts -O dtb -o $(DTB) $(DTS)

$(DTS) : $(SOPCINFO) $(BOARD_INFO)
    @which sopc2dts || (echo "sopc2dts not found.  Did you run
        embedded_command_shell.sh?"; exit 1)
    sopc2dts --input $(SOPCINFO) \
        --output $(DTS) \
        --type dts \
        --board $(BOARD_INFO) \
        --clocks

```

```
# ---------------------------------------------------------------------
# tar
# ---------------------------------------------------------------------
.phony: tar
tar: $(TARFILE)

$(TARFILE): $(TARFILES)
    tar zcfC $(TARFILE) .. $(TARFILES:%=final_project-hw/%)

# ---------------------------------------------------------------------
# clean
# ---------------------------------------------------------------------
.PHONY: clean quartus-clean qsys-clean project-clean dtb-clean rom-clean
clean: quartus-clean qsys-clean project-clean dtb-clean rom-clean

project-clean:
    rm -rf $(QPF) $(QSF) $(SDC)

qsys-clean:
    rm -rf $(SOPCINFO) $(QIP) $(SYSTEM)/ .qsys_edit \
    hps_isw_handoff/ hps_sdram_p0_summary.csv

quartus-clean:
    rm -rf $(SOF) $(RBF) output_files db incremental_db $(SYSTEM).qdf \
    c5_pin_model_dump.txt $(HPS_PIN_MAP)

dtb-clean:
    rm -rf $(DTS) $(DTB)

rom-clean:
    rm -f $(ROM_FILES)
```

A.7 Hardware: soc_system.tcl (Quartus project bring-up)

```
# 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]
```

```
116     PIN_AH18 GPIO_0[10]
117     PIN_AH17 GPIO_0[11]
118     PIN_AG16 GPIO_0[12]
119     PIN_AE16 GPIO_0[13]
120     PIN_AF16 GPIO_0[14]
121     PIN_AG17 GPIO_0[15]
122     PIN_AA18 GPIO_0[16]
123     PIN_AA19 GPIO_0[17]
124     PIN_AE17 GPIO_0[18]
125     PIN_AC20 GPIO_0[19]
126     PIN_AH19 GPIO_0[20]
127     PIN_AJ20 GPIO_0[21]
128     PIN_AH20 GPIO_0[22]
129     PIN_AK21 GPIO_0[23]
130     PIN_AD19 GPIO_0[24]
131     PIN_AD20 GPIO_0[25]
132     PIN_AE18 GPIO_0[26]
133     PIN_AE19 GPIO_0[27]
134     PIN_AF20 GPIO_0[28]
135     PIN_AF21 GPIO_0[29]
136     PIN_AF19 GPIO_0[30]
137     PIN_AG21 GPIO_0[31]
138     PIN_AF18 GPIO_0[32]
139     PIN_AG20 GPIO_0[33]
140     PIN_AG18 GPIO_0[34]
141     PIN_AJ21 GPIO_0[35]
142
143     PIN_AB17 GPIO_1[0]
144     PIN_AA21 GPIO_1[1]
145     PIN_AB21 GPIO_1[2]
146     PIN_AC23 GPIO_1[3]
147     PIN_AD24 GPIO_1[4]
148     PIN_AE23 GPIO_1[5]
149     PIN_AE24 GPIO_1[6]
150     PIN_AF25 GPIO_1[7]
151     PIN_AF26 GPIO_1[8]
152     PIN_AG25 GPIO_1[9]
153     PIN_AG26 GPIO_1[10]
154     PIN_AH24 GPIO_1[11]
155     PIN_AH27 GPIO_1[12]
156     PIN_AJ27 GPIO_1[13]
157     PIN_AK29 GPIO_1[14]
158     PIN_AK28 GPIO_1[15]
159     PIN_AK27 GPIO_1[16]
160     PIN_AJ26 GPIO_1[17]
161     PIN_AK26 GPIO_1[18]
162     PIN_AH25 GPIO_1[19]
163     PIN_AJ25 GPIO_1[20]
164     PIN_AJ24 GPIO_1[21]
165     PIN_AK24 GPIO_1[22]
166     PIN_AG23 GPIO_1[23]
167     PIN_AK23 GPIO_1[24]
168     PIN_AH23 GPIO_1[25]
169     PIN_AK22 GPIO_1[26]
170     PIN_AJ22 GPIO_1[27]
171     PIN_AH22 GPIO_1[28]
172     PIN_AG22 GPIO_1[29]
173     PIN_AF24 GPIO_1[30]
174     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]
```

```
293
294         PIN_B13 VGA_B[0]
295         PIN_G13 VGA_B[1]
296         PIN_H13 VGA_B[2]
297         PIN_F14 VGA_B[3]
298         PIN_H14 VGA_B[4]
299         PIN_F15 VGA_B[5]
300         PIN_G15 VGA_B[6]
301         PIN_J14 VGA_B[7]
302
303         PIN_A11 VGA_CLK
304         PIN_B11 VGA_HS
305         PIN_D11 VGA_VS
306         PIN_F10 VGA_BLANK_N
307         PIN_C10 VGA_SYNC_N
308 } {
309         set_location_assignment $pin -to $port
310         set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
311 }
312
313 # HPS assignments
314
315 # 3.3-V LVTTL pins
316 foreach port {
317         HPS_CONV_USB_N
318         HPS_ENET_GTX_CLK
319         HPS_ENET_INT_N
320         HPS_ENET_MDC
321         HPS_ENET_MDIO
322         HPS_ENET_RX_CLK
323         HPS_ENET_RX_DATA[0]
324         HPS_ENET_RX_DATA[1]
325         HPS_ENET_RX_DATA[2]
326         HPS_ENET_RX_DATA[3]
327         HPS_ENET_RX_DV
328         HPS_ENET_TX_DATA[0]
329         HPS_ENET_TX_DATA[1]
330         HPS_ENET_TX_DATA[2]
331         HPS_ENET_TX_DATA[3]
332         HPS_ENET_TX_EN
333         HPS_GSENSOR_INT
334         HPS_I2C1_SCLK
335         HPS_I2C1_SDAT
336         HPS_I2C2_SCLK
337         HPS_I2C2_SDAT
338         HPS_I2C_CONTROL
339         HPS_KEY
340         HPS_LED
341         HPS_LTC_GPIO
342         HPS_SD_CLK
343         HPS_SD_CMD
344         HPS_SD_DATA[0]
345         HPS_SD_DATA[1]
346         HPS_SD_DATA[2]
347         HPS_SD_DATA[3]
348         HPS_SPIM_CLK
349         HPS_SPIM_MISO
350         HPS_SPIM_MOSI
351         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
```

A.8 Software: game.h

```
/*
 * game.h  -  shared interface between the kernel module and the
 * user-space game application for the Maze Chase Game.
 *
 * Layout follows lab3/lab3-sw/vga_ball.h: ioctl numbers plus the
 * argument structs that travel through them.
 */

#ifndef _GAME_H
#define _GAME_H

#include <linux/ioctl.h>

#ifndef __KERNEL__
#include <stdint.h>
#else
#include <linux/types.h>
typedef __u8  uint8_t;
typedef __u32 uint32_t;
#endif

/* ------------------------------------------------------------------ */
/* Game-logic constants (mirror the Verilog parameters)               */
/* ------------------------------------------------------------------ */
#define GAME_MAZE_COLS  30
#define GAME_MAZE_ROWS  20
#define GAME_NUM_TILES   6

#define GAME_CELL_WALL     0
#define GAME_CELL_PATH     1
#define GAME_CELL_SPECIAL  2
#define GAME_CELL_GATE     3

/* GAME_STATUS bit positions (also the bit layout of the FPGA register) */
#define GAME_BIT_GATE_OPEN   0x1
#define GAME_BIT_EVADER_WIN  0x2
#define GAME_BIT_GHOST_WIN   0x4
#define GAME_BIT_RESET       0x8

/* ------------------------------------------------------------------ */
/* Position type used by the ioctls                                   */
/* ------------------------------------------------------------------ */
typedef struct {
    uint8_t col;     /* 0 .. 29 */
    uint8_t row;     /* 0 .. 19 */
} game_pos_t;

/* Snapshot of the writable portion of the game state.  The user app
 * sends this once per movement tick; the driver issues three 32-bit
 * writes to the EVADER_POS / GHOST0_POS / GHOST1_POS registers, plus
 * one optional write to GAME_STATUS.                                 */
typedef struct {
    game_pos_t evader;
    game_pos_t ghost0;
    game_pos_t ghost1;
    uint32_t   game_status;     /* see GAME_BIT_* above               */
```

```
    uint32_t   write_status;    /* 0 = leave GAME_STATUS untouched,
                                   nonzero = write game_status field    */
} game_write_t;

/* Snapshot of everything the FPGA reports back to the application.    */
typedef struct {
    uint32_t tile_status;               /* bits[5:0] activated flags  */
    uint32_t tile_time[GAME_NUM_TILES]; /* per-tile frame counts      */
    uint32_t game_status;               /* current GAME_STATUS reg    */
    uint32_t frame_count;               /* rolling 16-bit frame ctr   */
} game_read_t;

/* ------------------------------------------------------------------- */
/* ioctls                                                               */
/* ------------------------------------------------------------------- */
#define GAME_MAGIC 'g'

#define GAME_WRITE_STATE _IOW (GAME_MAGIC, 1, game_write_t)
#define GAME_READ_STATE  _IOR (GAME_MAGIC, 2, game_read_t )
#define GAME_RESET_GAME  _IO  (GAME_MAGIC, 3)
#define GAME_WAIT_VSYNC  _IOR (GAME_MAGIC, 4, uint32_t)

#endif /* _GAME_H */
```

A.9 Software: game.c (kernel module)

```c
/*
 * game.c  -  Linux kernel module for the Maze Chase Game peripheral.
 *
 * Modelled on lab3's vga_ball.c.  Registers a misc character device
 * called "game" and exposes four ioctls that read or write the 32-bit
 * registers the FPGA exports at the Lightweight HPS-to-FPGA bridge
 * (physical 0xFF200000 on the DE1-SoC).
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.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 "game.h"

#define DRIVER_NAME "game"

/* Byte offsets of each 32-bit register inside the BAR. */
#define REG_EVADER_POS   0x00
#define REG_GHOST0_POS   0x04
#define REG_GHOST1_POS   0x08
#define REG_TILE_STATUS  0x0C
#define REG_TILE0_TIME   0x10
#define REG_TILE1_TIME   0x14
#define REG_TILE2_TIME   0x18
#define REG_TILE3_TIME   0x1C
#define REG_TILE4_TIME   0x20
#define REG_TILE5_TIME   0x24
#define REG_GAME_STATUS  0x28
#define REG_FRAME_SYNC   0x2C

#define ENCODE_POS(col, row) ((((uint32_t)(row) & 0x1F) << 5) | \
                              ((uint32_t)(col) & 0x1F))

/*
 * Information about our device
 */
struct game_dev {
    struct resource res;
    void __iomem  *virtbase;
    game_write_t   last_write;
} dev;

/* ---------------------------------------------------------------- */
/* Low-level register helpers                                       */
/* ---------------------------------------------------------------- */
```

```
57 static inline void game_write_pos(uint32_t off, game_pos_t p)
58 {
59     iowrite32(ENCODE_POS(p.col, p.row), dev.virtbase + off);
60 }
61
62 static long do_write_state(const game_write_t *w)
63 {
64     game_write_pos(REG_EVADER_POS, w->evader);
65     game_write_pos(REG_GHOST0_POS, w->ghost0);
66     game_write_pos(REG_GHOST1_POS, w->ghost1);
67     if (w->write_status)
68         iowrite32(w->game_status & 0xF, dev.virtbase + REG_GAME_STATUS);
69     dev.last_write = *w;
70     return 0;
71 }
72
73 static long do_read_state(game_read_t *r)
74 {
75     int i;
76     r->tile_status = ioread32(dev.virtbase + REG_TILE_STATUS);
77     for (i = 0; i < GAME_NUM_TILES; i++)
78         r->tile_time[i] = ioread32(dev.virtbase + REG_TILE0_TIME + 4 * i);
79     r->game_status = ioread32(dev.virtbase + REG_GAME_STATUS);
80     r->frame_count = ioread32(dev.virtbase + REG_FRAME_SYNC) & 0xFFFF;
81     return 0;
82 }
83
84 static long do_reset_game(void)
85 {
86     /* GAME_RESET self-clears in hardware.  Just pulse the bit. */
87     iowrite32(GAME_BIT_RESET, dev.virtbase + REG_GAME_STATUS);
88     return 0;
89 }
90
91 static long do_wait_vsync(uint32_t *out_frame)
92 {
93     uint32_t f0 = ioread32(dev.virtbase + REG_FRAME_SYNC) & 0xFFFF;
94     uint32_t f1 = f0;
95     int spins = 0;
96     /* Busy-poll until FRAME_COUNT changes.  Worst case ~16.7 ms. */
97     while (f1 == f0) {
98         if ((++spins & 0xFFF) == 0)
99             cond_resched();
100         f1 = ioread32(dev.virtbase + REG_FRAME_SYNC) & 0xFFFF;
101     }
102     *out_frame = f1;
103     return 0;
104 }
105
106 /* ------------------------------------------------------------------ */
107 /* ioctl dispatch                                                     */
108 /* ------------------------------------------------------------------ */
109 static long game_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
110 {
111     game_write_t w;
112     game_read_t  r;
113     uint32_t     frame;
114     long         ret;
115
```

```
    switch (cmd) {
    case GAME_WRITE_STATE:
        if (copy_from_user(&w, (void __user *)arg, sizeof(w)))
            return -EFAULT;
        ret = do_write_state(&w);
        break;

    case GAME_READ_STATE:
        ret = do_read_state(&r);
        if (ret == 0 && copy_to_user((void __user *)arg, &r, sizeof(r)))
            return -EFAULT;
        break;

    case GAME_RESET_GAME:
        ret = do_reset_game();
        break;

    case GAME_WAIT_VSYNC:
        ret = do_wait_vsync(&frame);
        if (ret == 0 && copy_to_user((void __user *)arg, &frame,
                                     sizeof(frame)))
            return -EFAULT;
        break;

    default:
        return -EINVAL;
    }
    return ret;
}

static const struct file_operations game_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = game_ioctl,
};

static struct miscdevice game_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name  = DRIVER_NAME,
    .fops  = &game_fops,
};

/* ------------------------------------------------------------------ */
/* Platform driver bind/unbind                                        */
/* ------------------------------------------------------------------ */
static int __init game_probe(struct platform_device *pdev)
{
    int ret;

    ret = misc_register(&game_misc_device);
    if (ret) {
        pr_err(DRIVER_NAME ": misc_register failed\n");
        return ret;
    }

    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }
```

```
175
176     if (request_mem_region(dev.res.start, resource_size(&dev.res),
177                            DRIVER_NAME) == NULL) {
178         ret = -EBUSY;
179         goto out_deregister;
180     }
181
182     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
183     if (dev.virtbase == NULL) {
184         ret = -ENOMEM;
185         goto out_release_mem_region;
186     }
187
188     /* Make sure the game is in a known state at module load. */
189     iowrite32(GAME_BIT_RESET, dev.virtbase + REG_GAME_STATUS);
190
191     pr_info(DRIVER_NAME ": probed at 0x%llx (size=%llu)\n",
192             (unsigned long long)dev.res.start,
193             (unsigned long long)resource_size(&dev.res));
194     return 0;
195
196 out_release_mem_region:
197     release_mem_region(dev.res.start, resource_size(&dev.res));
198 out_deregister:
199     misc_deregister(&game_misc_device);
200     return ret;
201 }
202
203 static int game_remove(struct platform_device *pdev)
204 {
205     iounmap(dev.virtbase);
206     release_mem_region(dev.res.start, resource_size(&dev.res));
207     misc_deregister(&game_misc_device);
208     return 0;
209 }
210
211 #ifdef CONFIG_OF
212 static const struct of_device_id game_of_match[] = {
213     { .compatible = "csee4840,game_peripheral-1.0" },
214     {},
215 };
216 MODULE_DEVICE_TABLE(of, game_of_match);
217 #endif
218
219 static struct platform_driver game_driver = {
220     .driver = {
221         .name           = DRIVER_NAME,
222         .owner          = THIS_MODULE,
223         .of_match_table = of_match_ptr(game_of_match),
224     },
225     .remove = __exit_p(game_remove),
226 };
227
228 static int __init game_init(void)
229 {
230     pr_info(DRIVER_NAME ": init\n");
231     return platform_driver_probe(&game_driver, game_probe);
232 }
233
```

```
234 static void __exit game_exit(void)
235 {
236     platform_driver_unregister(&game_driver);
237     pr_info(DRIVER_NAME ": exit\n");
238 }
239 
240 module_init(game_init);
241 module_exit(game_exit);
242 
243 MODULE_LICENSE("GPL");
244 MODULE_AUTHOR("CSEE 4840 Final Project");
245 MODULE_DESCRIPTION("Maze Chase Game peripheral driver");
```

A.10 Software: game_app.c (user-space game loop)

```c
/*
 * game_app.c  -  HPS-side game loop for the Maze Chase Game.
 *
 * Drives the FPGA peripheral through the four /dev/game ioctls
 * declared in game.h.  Responsibilities:
 *   - frame pacing at 60 Hz (GAME_WAIT_VSYNC)
 *   - movement tick at 5 Hz (every 12 frames)
 *   - USB keyboard read on a dedicated pthread (libusb)
 *   - BFS shortest-path AI for both ghosts (over a 30x20 grid)
 *   - win/loss bookkeeping and ENTER-to-restart
 *
 * Build: `make` in this directory.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "game.h"
#include "maze_data.h"
#include "usbkeyboard.h"

/* ------------------------------------------------------------------ */
/* Constants                                                          */
/* ------------------------------------------------------------------ */
#define DEV_GAME       "/dev/game"
#define FRAMES_PER_TICK 12   /* 60 Hz / 12 = 5 Hz evader movement       */
#define GHOST_TICKS_PER_MOVE 2 /* ghosts move every Nth evader tick      */

/* USB HID Usage IDs (Keyboard page 0x07) - identical to lab2 mapping. */
#define HID_KEY_ENTER  0x28
#define HID_KEY_ESC    0x29
#define HID_KEY_RIGHT  0x4F
#define HID_KEY_LEFT   0x50
#define HID_KEY_DOWN   0x51
#define HID_KEY_UP     0x52

enum { DIR_NONE = 0, DIR_UP, DIR_DOWN, DIR_LEFT, DIR_RIGHT };

static const game_pos_t INITIAL_EVADER = { .col = 15, .row = 10 };
static const game_pos_t INITIAL_GHOST0 = { .col = 1,  .row = 2  };
static const game_pos_t INITIAL_GHOST1 = { .col = 28, .row = 2  };
static const game_pos_t GATE_POS       = { .col = 15, .row = 1  };

/* ------------------------------------------------------------------ */
/* Globals                                                            */
/* ------------------------------------------------------------------ */
static int g_game_fd  = -1;
```

```
57
58  /* USB keyboard handle + endpoint, plus background reader thread. */
59  static struct libusb_device_handle *g_keyboard = NULL;
60  static uint8_t                      g_kbd_endpoint;
61  static pthread_t                    g_kbd_thread;
62  static volatile int                 g_kbd_stop = 0;
63  static volatile int                 g_kbd_thread_running = 0;
64
65  /* These are touched by the keyboard thread AND the main loop, so
66   * protect them with a mutex.                                      */
67  static pthread_mutex_t g_input_mutex = PTHREAD_MUTEX_INITIALIZER;
68  static int g_held_dir = DIR_NONE;       /* shared - latest arrow held */
69  static int g_enter_pressed = 0;         /* shared - sticky until consumed */
70  static int g_esc_pressed = 0;           /* shared - sticky until consumed */
71
72  static game_pos_t g_evader, g_ghost0, g_ghost1;
73  static int        g_gate_open = 0;
74  static int        g_game_over = 0;
75  static int        g_game_won  = 0;
76
77  /* ------------------------------------------------------------------ */
78  /* Helpers                                                             */
79  /* ------------------------------------------------------------------ */
80  static int maze_passable(int col, int row)
81  {
82      if (col < 0 || col >= GAME_MAZE_COLS) return 0;
83      if (row < 0 || row >= GAME_MAZE_ROWS) return 0;
84      unsigned char c = maze_data[row][col];
85      return c != GAME_CELL_WALL;
86  }
87
88  /* The ghost should not be allowed to use the gate as a shortcut. */
89  static int maze_passable_ghost(int col, int row)
90  {
91      if (!maze_passable(col, row)) return 0;
92      if (col == GATE_POS.col && row == GATE_POS.row) return 0;
93      return 1;
94  }
95
96  static int pos_eq(game_pos_t a, game_pos_t b)
97  {
98      return a.col == b.col && a.row == b.row;
99  }
100
101 static int popcount6(uint32_t v)
102 {
103     int n = 0;
104     for (int i = 0; i < 6; i++) if (v & (1u << i)) n++;
105     return n;
106 }
107
108 /* ------------------------------------------------------------------ */
109 /* Driver wrappers                                                     */
110 /* ------------------------------------------------------------------ */
111 static void send_state(uint32_t status, int write_status)
112 {
113     game_write_t w = {
114         .evader        = g_evader,
115         .ghost0        = g_ghost0,
```

```
        .ghost1        = g_ghost1,
        .game_status   = status,
        .write_status  = write_status,
    };
    if (ioctl(g_game_fd, GAME_WRITE_STATE, &w) < 0)
        perror("ioctl(GAME_WRITE_STATE)");
}

static void read_state(game_read_t *r)
{
    if (ioctl(g_game_fd, GAME_READ_STATE, r) < 0)
        perror("ioctl(GAME_READ_STATE)");
}

static void wait_vsync(void)
{
    uint32_t frame;
    if (ioctl(g_game_fd, GAME_WAIT_VSYNC, &frame) < 0)
        perror("ioctl(GAME_WAIT_VSYNC)");
}

static void reset_game_hw(void)
{
    if (ioctl(g_game_fd, GAME_RESET_GAME) < 0)
        perror("ioctl(GAME_RESET_GAME)");
}

/* ------------------------------------------------------------------ */
/* USB keyboard input via libusb (lab2-style direct HID).
 *
 * The Linux input/evdev path proved unreliable on the DE1-SoC because
 * the kernel HID driver doesn't always bind cleanly.  Instead we claim
 * the HID interface ourselves with libusb and read raw 8-byte HID
 * reports.  Each report contains the modifier byte and up to 6 keycodes
 * that are CURRENTLY held; we translate the first arrow we see into
 * g_held_dir and watch for ENTER / ESC edges.
 *
 * The interrupt transfer is blocking, so it runs in its own pthread.
 * ------------------------------------------------------------------ */
static int kbd_packet_has(const struct usb_keyboard_packet *p, uint8_t code)
{
    int i;
    for (i = 0; i < 6; i++)
        if (p->keycode[i] == code) return 1;
    return 0;
}

static void *keyboard_thread_fn(void *arg)
{
    struct usb_keyboard_packet packet;
    struct usb_keyboard_packet prev = {0};
    int transferred;

    (void)arg;

    while (!g_kbd_stop) {
        /* 200 ms timeout so we can periodically check g_kbd_stop and
         * exit cleanly without relying on pthread_cancel.            */
        int r = libusb_interrupt_transfer(g_keyboard, g_kbd_endpoint,
```

```
                                        (unsigned char *)&packet,
                                        sizeof(packet),
                                        &transferred, 200);
        if (r == LIBUSB_ERROR_TIMEOUT) continue;
        if (r == LIBUSB_ERROR_NO_DEVICE || r == LIBUSB_ERROR_INTERRUPTED)
            break;
        if (r != 0) {
            fprintf(stderr, "libusb_interrupt_transfer: %s\n",
                    libusb_error_name(r));
            continue;
        }
        if (transferred != sizeof(packet)) continue;

        /* Determine the latest held arrow direction.  If the user holds
         * multiple arrows the priority is up>down>left>right.          */
        int dir = DIR_NONE;
        if      (kbd_packet_has(&packet, HID_KEY_UP   )) dir = DIR_UP;
        else if (kbd_packet_has(&packet, HID_KEY_DOWN )) dir = DIR_DOWN;
        else if (kbd_packet_has(&packet, HID_KEY_LEFT )) dir = DIR_LEFT;
        else if (kbd_packet_has(&packet, HID_KEY_RIGHT)) dir = DIR_RIGHT;

        /* ENTER / ESC are edge-triggered: only on a fresh press. */
        int enter_edge =  kbd_packet_has(&packet, HID_KEY_ENTER) &&
                         !kbd_packet_has(&prev,   HID_KEY_ENTER);
        int esc_edge   =  kbd_packet_has(&packet, HID_KEY_ESC  ) &&
                         !kbd_packet_has(&prev,   HID_KEY_ESC  );

        pthread_mutex_lock(&g_input_mutex);
        g_held_dir = dir;
        if (enter_edge) g_enter_pressed = 1;
        if (esc_edge)   g_esc_pressed   = 1;
        pthread_mutex_unlock(&g_input_mutex);

        prev = packet;
    }
    g_kbd_thread_running = 0;
    return NULL;
}

/* Snapshot the keyboard state into local variables under the mutex so
 * the rest of the main loop can use the existing g_held_dir-style
 * variables without locking.  Also clears the sticky ENTER flag.     */
static void poll_input(void)
{
    pthread_mutex_lock(&g_input_mutex);
    /* g_held_dir is already up to date in shared memory; nothing to
     * copy out.  We only need to *consume* the ENTER edge here so the
     * caller sees a single-frame pulse.                              */
    pthread_mutex_unlock(&g_input_mutex);
}

/* ------------------------------------------------------------------ */
/* Movement and AI                                                    */
/* ------------------------------------------------------------------ */
static void move_evader(void)
{
    int dc = 0, dr = 0;
    int dir;

```

```
    pthread_mutex_lock(&g_input_mutex);
    dir = g_held_dir;
    pthread_mutex_unlock(&g_input_mutex);

    switch (dir) {
    case DIR_UP:    dr = -1; break;
    case DIR_DOWN:  dr =  1; break;
    case DIR_LEFT:  dc = -1; break;
    case DIR_RIGHT: dc =  1; break;
    default: return;
    }
    int nc = g_evader.col + dc;
    int nr = g_evader.row + dr;
    if (!maze_passable(nc, nr)) return;
    /* The gate cell is only passable for the Evader after the gate is
     * unlocked (cell type GATE).                                        */
    if (nr == GATE_POS.row && nc == GATE_POS.col && !g_gate_open) return;

    g_evader.col = (uint8_t)nc;
    g_evader.row = (uint8_t)nr;
}

/* ------------- BFS over the 30x20 grid ----------------------------- */
/* We re-use a single static workspace to avoid heap pressure.          */
#define BFS_N (GAME_MAZE_ROWS * GAME_MAZE_COLS)

static uint8_t  bfs_visited[BFS_N];
static int16_t  bfs_parent [BFS_N];
static uint16_t bfs_queue  [BFS_N];

static inline int tile_idx(int col, int row)
{
    return row * GAME_MAZE_COLS + col;
}

/* Compute the next tile a ghost should move into to chase the Evader.
 * Returns the source tile (no movement) if no path exists.             */
static game_pos_t bfs_next_step(game_pos_t from, game_pos_t to)
{
    int src = tile_idx(from.col, from.row);
    int dst = tile_idx(to.col,   to.row  );

    memset(bfs_visited, 0, sizeof(bfs_visited));
    for (int i = 0; i < BFS_N; i++) bfs_parent[i] = -1;

    int qh = 0, qt = 0;
    bfs_queue[qt++] = (uint16_t)src;
    bfs_visited[src] = 1;

    static const int dC[4] = {  0,  0, -1,  1 };
    static const int dR[4] = { -1,  1,  0,  0 };

    int found = 0;
    while (qh < qt && !found) {
        int cur = bfs_queue[qh++];
        int cc = cur % GAME_MAZE_COLS;
        int cr = cur / GAME_MAZE_COLS;
        for (int k = 0; k < 4; k++) {
            int nc = cc + dC[k];
```

```
            int nr = cr + dR[k];
            if (!maze_passable_ghost(nc, nr)) continue;
            int nidx = tile_idx(nc, nr);
            if (bfs_visited[nidx]) continue;
            bfs_visited[nidx] = 1;
            bfs_parent [nidx] = (int16_t)cur;
            if (nidx == dst) { found = 1; break; }
            bfs_queue[qt++] = (uint16_t)nidx;
        }
    }

    if (!found || bfs_parent[dst] < 0) {
        return from;  /* No path - hold position */
    }

    /* Walk back from dst until we find the tile whose parent is src.   */
    int cur = dst;
    while (bfs_parent[cur] != src && bfs_parent[cur] >= 0)
        cur = bfs_parent[cur];

    game_pos_t out;
    out.col = (uint8_t)(cur % GAME_MAZE_COLS);
    out.row = (uint8_t)(cur / GAME_MAZE_COLS);
    return out;
}

static void move_ghosts(void)
{
    g_ghost0 = bfs_next_step(g_ghost0, g_evader);
    g_ghost1 = bfs_next_step(g_ghost1, g_evader);
}

/* ---------------------------------------------------------------------- */
/* Game-state housekeeping                                                */
/* ---------------------------------------------------------------------- */
static void reset_logical_state(void)
{
    g_evader = INITIAL_EVADER;
    g_ghost0 = INITIAL_GHOST0;
    g_ghost1 = INITIAL_GHOST1;
    g_gate_open = 0;
    g_game_over = 0;
    g_game_won  = 0;
}

static void check_gate_unlock(uint32_t tile_status)
{
    if (!g_gate_open && popcount6(tile_status) >= 3) {
        g_gate_open = 1;
        send_state(GAME_BIT_GATE_OPEN, 1);
        printf("Gate unlocked!\n");
    }
}

static void check_win_loss(void)
{
    if (g_game_over) return;
    if (pos_eq(g_evader, g_ghost0) || pos_eq(g_evader, g_ghost1)) {
        g_game_over = 1;
```

```
352         g_game_won  = 0;
353         send_state(GAME_BIT_GHOST_WIN | (g_gate_open ? GAME_BIT_GATE_OPEN : 0),
                1);
354         printf("Ghost catches Evader - you lose.  Press ENTER to restart.\n");
355         return;
356     }
357     if (g_gate_open && pos_eq(g_evader, GATE_POS)) {
358         g_game_over = 1;
359         g_game_won  = 1;
360         send_state(GAME_BIT_EVADER_WIN | GAME_BIT_GATE_OPEN, 1);
361         printf("Evader reaches gate - you win!  Press ENTER to restart.\n");
362     }
363 }
364
365 /* ---------------------------------------------------------------------- */
366 /* Main                                                                   */
367 /* ---------------------------------------------------------------------- */
368 /* Atomically read-and-clear the ENTER edge flag. */
369 static int consume_enter(void)
370 {
371     int v;
372     pthread_mutex_lock(&g_input_mutex);
373     v = g_enter_pressed;
374     g_enter_pressed = 0;
375     pthread_mutex_unlock(&g_input_mutex);
376     return v;
377 }
378
379 /* Read the ESC edge flag (latched - never cleared, used for graceful exit). */
380 static int peek_esc(void)
381 {
382     int v;
383     pthread_mutex_lock(&g_input_mutex);
384     v = g_esc_pressed;
385     pthread_mutex_unlock(&g_input_mutex);
386     return v;
387 }
388
389 int main(void)
390 {
391     printf("Maze Chase Game starting...\n");
392
393     g_game_fd = open(DEV_GAME, O_RDWR);
394     if (g_game_fd < 0) {
395         fprintf(stderr, "could not open %s: %s\n", DEV_GAME, strerror(errno));
396         return 1;
397     }
398
399     /* Open the USB keyboard via libusb (claims the HID interface). */
400     g_keyboard = openkeyboard(&g_kbd_endpoint);
401     if (g_keyboard == NULL) {
402         fprintf(stderr, "warning: no USB keyboard found - controls disabled\n");
403     } else {
404         printf("USB keyboard claimed (endpoint 0x%02x)\n", g_kbd_endpoint);
405         g_kbd_thread_running = 1;
406         if (pthread_create(&g_kbd_thread, NULL, keyboard_thread_fn, NULL) != 0) {
407             perror("pthread_create(keyboard)");
408             g_kbd_thread_running = 0;
409         }
```

```
410     }
411
412     reset_game_hw();
413     reset_logical_state();
414     send_state(0, 1);   /* clear all status bits */
415
416     int move_tick = 0;
417     int ghost_tick = 0;
418
419     while (!peek_esc()) {
420         wait_vsync();
421         poll_input();
422
423         if (g_game_over) {
424             if (consume_enter()) {
425                 printf("Restarting...\n");
426                 reset_game_hw();
427                 reset_logical_state();
428                 send_state(0, 1);
429                 move_tick = 0;
430                 ghost_tick = 0;
431             }
432             continue;
433         }
434         (void)consume_enter();   /* discard ENTER during play */
435
436         /* Read tile_status every frame for gate-unlock check */
437         game_read_t st;
438         read_state(&st);
439         check_gate_unlock(st.tile_status);
440
441         if (++move_tick >= FRAMES_PER_TICK) {
442             move_tick = 0;
443             move_evader();
444             /* Ghosts step on only every GHOST_TICKS_PER_MOVE-th evader
445              * tick so they move slower than the player.                */
446             if (++ghost_tick >= GHOST_TICKS_PER_MOVE) {
447                 ghost_tick = 0;
448                 move_ghosts();
449             }
450             send_state(0, 0);   /* push new positions, leave status */
451             check_win_loss();
452         }
453     }
454
455     printf("Shutting down...\n");
456     if (g_keyboard) {
457         g_kbd_stop = 1;
458         if (g_kbd_thread_running) pthread_join(g_kbd_thread, NULL);
459         libusb_close(g_keyboard);
460         libusb_exit(NULL);
461     }
462     close(g_game_fd);
463     return 0;
464 }
```

A.11 Software: usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

A.12 Software: usbkeyboard.c

```c
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * https://www.usb.org/sites/default/files/documents/hid1_11.pdf
 * https://usb.org/sites/default/files/hut1_5.pdf
 */

struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  if (libusb_init(NULL) < 0) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
  }

  for (d = 0; d < num_devs; d++) {
    libusb_device *dev = devs[d];
    if (libusb_get_device_descriptor(dev, &desc) < 0) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0; i < config->bNumInterfaces; i++)
        for (k = 0; k < config->interface[i].num_altsetting; k++) {
          const struct libusb_interface_descriptor *inter =
              config->interface[i].altsetting + k;
          if (inter->bInterfaceClass == LIBUSB_CLASS_HID &&
              inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
            int r;
            if ((r = libusb_open(dev, &keyboard)) != 0) {
              fprintf(stderr, "Error: libusb_open failed: %d\n", r);
              exit(1);
            }
            if (libusb_kernel_driver_active(keyboard, i))
              libusb_detach_kernel_driver(keyboard, i);
            libusb_set_auto_detach_kernel_driver(keyboard, i);
            if ((r = libusb_claim_interface(keyboard, i)) != 0) {
              fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
              exit(1);
```

```
57              }
58              *endpoint_address = inter->endpoint[0].bEndpointAddress;
59              goto found;
60            }
61          }
62      }
63    }
64
65   found:
66    libusb_free_device_list(devs, 1);
67
68    return keyboard;
69  }
```

A.13 Software: Makefile

```
# ============================================================================
# Software Makefile - Maze Chase Game
#
# Builds (lab3 pattern):
#   game.ko     - kernel module that exposes /dev/game
#   game_app    - user-space binary that runs the game loop
# ============================================================================

ifneq (${KERNELRELEASE},)

# We are being included from the kernel build system.
        obj-m := game.o

else

# Standalone make: ARM cross-compile by default for DE1-SoC.
        KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
        PWD := $(shell pwd)

default: module game_app

module:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

game_app: game_app.c usbkeyboard.c usbkeyboard.h game.h maze_data.h
    $(CC) -O2 -Wall -Wextra -o $@ game_app.c usbkeyboard.c \
        -lusb-1.0 -lpthread

clean:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
    ${RM} game_app

TARFILES = Makefile README game.h game.c game_app.c \
       usbkeyboard.h usbkeyboard.c maze_data.h
TARFILE  = final_project-sw.tar.gz
.PHONY: tar
tar: $(TARFILE)

$(TARFILE): $(TARFILES)
    tar zcfC $(TARFILE) .. $(TARFILES:%=final_project-sw/%)

endif
```

A.14 Software: README

```
Maze Chase Game - HPS software (CSEE 4840 Final Project)
=========================================================

This directory contains the Linux device driver and user-space game
application for the Maze Chase Game peripheral.  The layout follows
lab3/lab3-sw exactly:

    game.h        - shared ioctl definitions (kernel + user)
    game.c        - kernel module (misc device /dev/game)
    game_app.c    - user-space game loop (BFS, USB input, etc.)
    maze_data.h   - auto-generated maze layout shared with the hardware
    Makefile      - builds module + app

`maze_data.h` is produced by ../final_project-hw/gen_mif.py; this
guarantees the FPGA Maze ROM and the HPS collision array describe the
exact same maze.

---------------------------------------------------------------------
Building
---------------------------------------------------------------------
On the DE1-SoC target:

    make                          # builds game.ko and game_app

Make sure the FPGA bitstream (../final_project-hw/output_files/soc_system.rbf)
has already been written to the SD card and that the corresponding
device tree blob has been installed; the kernel driver depends on the
"csee4840,game_peripheral-1.0" compatible node.

---------------------------------------------------------------------
Running
---------------------------------------------------------------------
    echo 8 > /proc/sys/kernel/printk    # enable kernel printks
    insmod game.ko
    dmesg | tail
    ls /dev/game                        # should exist
    ./game_app

A USB HID keyboard must be plugged into the DE1-SoC USB port.  Use
arrow keys to move the Evader.  Press ENTER after a win/loss screen to
restart.  Stop the application with Ctrl-C.

To remove the module:

    rmmod game

---------------------------------------------------------------------
ioctl reference (defined in game.h)
---------------------------------------------------------------------
    GAME_WRITE_STATE  - send new entity positions (and optionally
                        GAME_STATUS) to the FPGA registers.
    GAME_READ_STATE   - read TILE_STATUS, TILE*_TIME, GAME_STATUS,
                        FRAME_SYNC into a single struct.
    GAME_RESET_GAME   - pulse the self-clearing GAME_RESET bit.
    GAME_WAIT_VSYNC   - busy-wait inside the driver until the FPGA
                        frame counter advances (returns the new value).
```