Ultrasonic Fruit Ninja on DE1-SoC

Final Project Report

CSEE 4840 Embedded System Design – Spring 2026

Peiheng Li (pl2978) Chengrui Li (cl4750) Yitong Bai (yb2636)

Contents

Abstract

This report describes an embedded Fruit Ninja-style game implemented on a Terasic DE1-SoC board. The player interacts with the game by moving a hand near an HC-SR04 ultrasonic sensor, while the FPGA fabric produces VGA graphics and exposes game-control registers to the HPS processor. The hardware design includes an ultrasonic trigger/echo counter, a memory-mapped register file, a VGA timing generator, sprite and background ROMs, fixed-point sprite rotation, and on-screen HUD rendering. The HPS side runs Linux, a misc-device kernel driver, and a user-space C game loop that manages spawning, collision rules, score, lives, combo state, and hardware register updates. The implementation avoids a frame buffer; instead, each VGA pixel is composed directly from current game registers, ROM outputs, and HUD logic.

1 Introduction and Motivation

The project implements a compact hardware/software game system on the DE1-SoC. The design uses the HPS ARM processor for high-level game policy and the FPGA fabric for deterministic I/O and video generation. The resulting system demonstrates the main embedded-system design theme of the course: a hardware/software boundary selected according to timing, flexibility, and resource needs.

The project uses an HC-SR04 ultrasonic sensor rather than a camera. Repository documentation indicates that an earlier camera-based approach was considered, but the implemented repository uses only the ultrasonic sensor path. The ultrasonic interface gives one distance measurement at a time, which is sufficient for detecting a near-field swipe event. This choice also avoids camera frame buffering, lighting sensitivity, and a more complex image-processing pipeline.

1.1 Design Objectives and Scope

The implemented system has four concrete objectives. First, the player input path must be simple enough to run without a camera frame buffer or computer-vision pipeline. Second, the FPGA must drive a live VGA display directly from current game state. Third, the HPS must be able to change gameplay parameters and object state without re-synthesizing hardware. Fourth, the hardware/software interface must be small and explicit enough to debug with direct register reads.

The scope of the repository is therefore a complete embedded game prototype rather than a general game engine. It implements one ultrasonic gesture input, three object slots, a fixed 640 by 480 VGA output mode, a fixed set of sprite and tile assets, and a Linux ioctl interface. It does not implement audio, runtime asset loading, dynamic resolution changes, networking, or persistent high-score storage.

1.2 Implemented Gameplay

The implemented game is driven by `software/game.c`. Fruit and bomb objects spawn near the top of the VGA screen, rotate, drift horizontally, and move downward toward a horizontal cut line at `LINE_Y = 320`. A hand moving within `NEAR_CM = 20` cm of the ultrasonic sensor is interpreted as a swipe when the software observes a far-to-near transition in a new ultrasonic sample.

- Cutting a fruit increments the score and combo count, then displays the corresponding blown sprite for about one second.
- Cutting a bomb decrements lives and resets the combo display for that swipe.
- Missing a fruit decrements lives and resets the combo. Letting a bomb fall off-screen has no penalty.
- Every tenth consecutive fruit combo heals one life if the player is below the maximum of three lives.

- Levels are derived from score: Level 1 below 10 points, Level 2 from 10 points, and Level 3 from 20 points. The level controls the number of simultaneous active object slots.

1.3 End-to-End Execution Scenario

A typical frame begins with the HPS application polling the ultrasonic CSR through the driver. If the CSR reports a new sample or a changed sample ID, `game.c` converts the echo pulse width into an approximate distance. A transition from a far reading to a near reading is treated as one swipe. The application then updates each active object, evaluates whether a swipe intersects any object near the cut line, updates score/lives/combo state, and writes the resulting render state back through the driver. The FPGA continuously reads those registers while it scans out VGA pixels, so visual changes appear on the next relevant pixels without a software frame-buffer copy.

2 System Overview

The system contains three main domains: Linux software on the HPS, custom RTL in the FPGA fabric, and external peripherals. Figure 1 shows the implemented system-level organization.

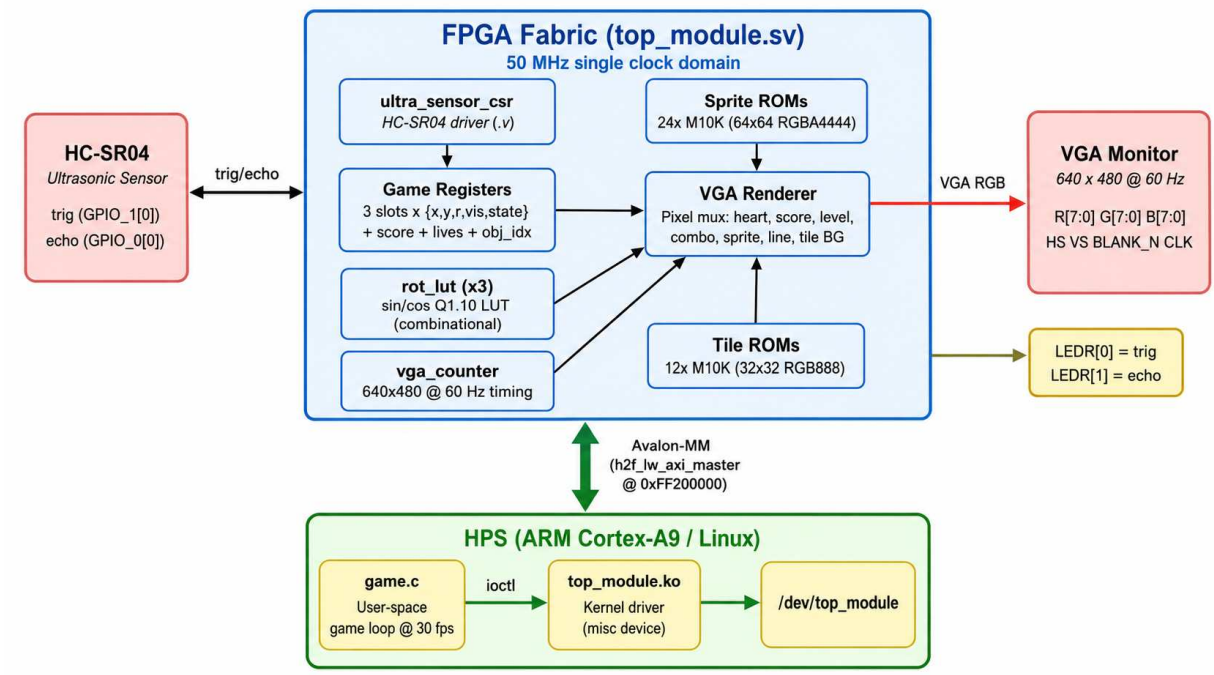

Figure 1: System overview showing HPS software, the lightweight HPS-to-FPGA bridge, custom FPGA logic, VGA output, LEDs, and the HC-SR04 sensor.

The HPS side is responsible for decisions that are easier to express and tune in C: object spawning, game rules, terminal logging, and parameter constants such as the near-field threshold. The FPGA side is responsible for periodic signal generation, measurement, and video timing. This split keeps the Linux side away from per-pixel graphics work, while also keeping the FPGA side away from gameplay policy that may change during development.

The data path is intentionally narrow. The HPS writes only object coordinates, visibility, packed sprite state, score, and lives. The FPGA returns only ultrasonic status and timing data. No frame data, sprite bitmap data, or background image data crosses the bridge during gameplay.

Table 1: Implemented feature summary

Feature	Location	Evidence in repository
Ultrasonic distance capture	FPGA	`rtl/ultra_sensor_csr.v`: trigger period counter, echo synchronizer, pulse-width counter, sample ID, W1C new-sample flag
Memory-mapped register interface	FPGA/HPS	`rtl/top_module.sv`, `software/top_module.h`, `software/top_module.c`: 12 word-addressed registers exposed at `0xFF200000`
Linux device abstraction	HPS kernel	`software/top_module.c`: platform driver, `/dev/top_module`, five ioctl commands
Game loop and rules	HPS user space	`software/game.c`: object spawning, level progression, swipe detection, scoring, lives, combos
VGA timing	FPGA	`rtl/vga_counter.sv`: 640 by 480 timing using a 50 MHz input and 25 MHz VGA clock
Sprite rendering	FPGA	`rtl/top_module.sv`, `rtl/sprite_rom.sv`, `pics/*.hex`: three object slots and eight sprite textures per slot
Background and HUD	FPGA	`rtl/top_module.sv`, `rtl/tile_rom_rgb888.sv`: tiled background, dashed cut line, score, level, combo text, hearts
Direct sensor test tools	HPS user space / shell	`tools/ultra_read.c`, `tools/ultra_sonic.sh`: direct `/dev/mem` or `devmem` access to ultrasonic registers

Table 2: Hardware/software responsibility split

Subsystem	HPS responsibility	FPGA responsibility
Input sensing	Poll samples through `MOTION_READ`; convert echo ticks to centimeters; detect far-to-near swipes	Generate ultrasonic trigger; synchronize echo; count echo pulse width; expose status, count, and sample ID
Game state	Maintain objects, score, lives, combo, level, and collision rules	Store current render registers for three object slots and global HUD state
Graphics	Select sprite IDs, positions, visibility, blow state, rotation angle, and combo values	Generate VGA timing and compose background, sprites, cut line, score, level, combo text, and hearts per pixel
Build and deployment	Build kernel module and game; load `top_module.ko`; run `game`	Synthesized as a Platform Designer custom peripheral in the DE1-SoC Quartus project

3 Hardware/RTL Design

3.1 Top-Level Custom Peripheral

The custom FPGA peripheral is `rtl/top_module.sv`. It has one 50 MHz clock input, an active-high reset, an Avalon-MM slave port, VGA outputs, ten debug LED outputs, and two ultrasonic sensor pins. In the Quartus/Platform Designer component file `DE1_SoC_GHRD/top_module_hw.tcl`, this module is declared as `top_module` version 1.0 with:

- `addressUnits = WORDS`;
- `readWaitTime = 1`;
- `writeWaitTime = 0`;
- a 32-bit Avalon-MM slave interface;
- conduit interfaces for VGA, ultrasonic sensor pins, and LEDs.

The top-level DE1-SoC wrapper connects `GPIO_0[0]` to the ultrasonic echo input and `GPIO_1[0]` to the ultrasonic trigger output. It also routes the custom LED conduit to `LEDR`.

3.2 Ultrasonic Sensor CSR

The HC-SR04 interface is implemented in `rtl/ultra_sensor_csr.v`. A 23-bit counter generates a trigger pulse approximately once every 100 ms. The implemented trigger condition is `period_cnt > 100` and `period_cnt < 5100`; therefore the trigger is high for counter values 101 through 5099, which is 4999 cycles or approximately 99.98 μs at 50 MHz.

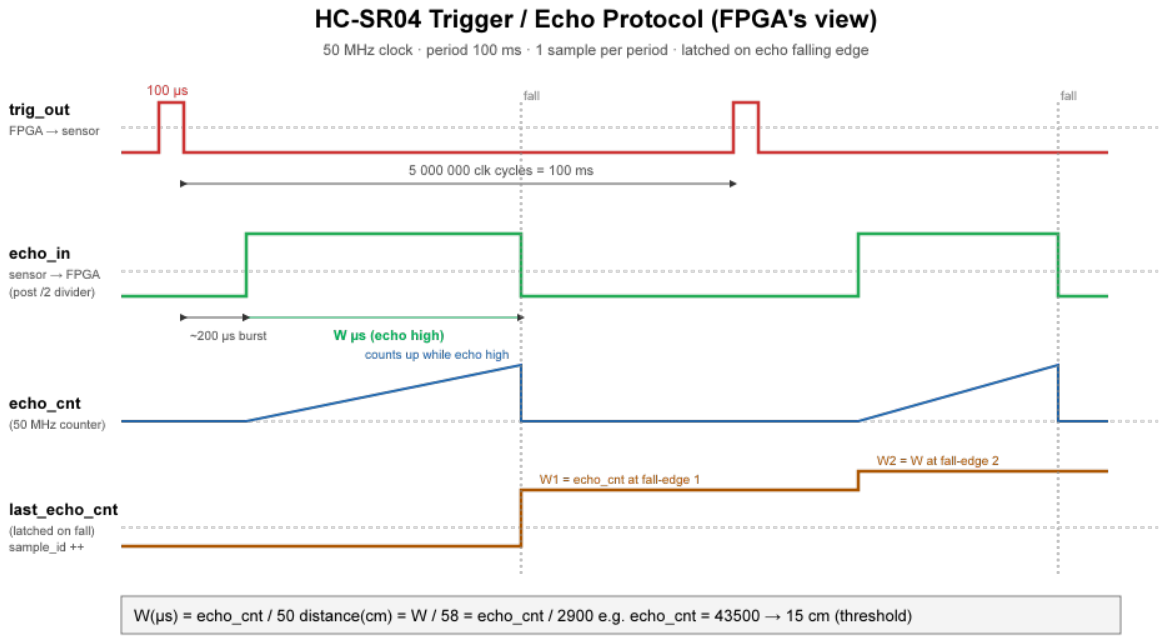

Figure 2: Ultrasonic trigger waveform for the HC-SR04 interface.

The echo input is synchronized through two flip-flops, and an additional delayed copy is used for edge detection. While synchronized echo is high, a 32-bit counter accumulates the echo pulse width in 50 MHz

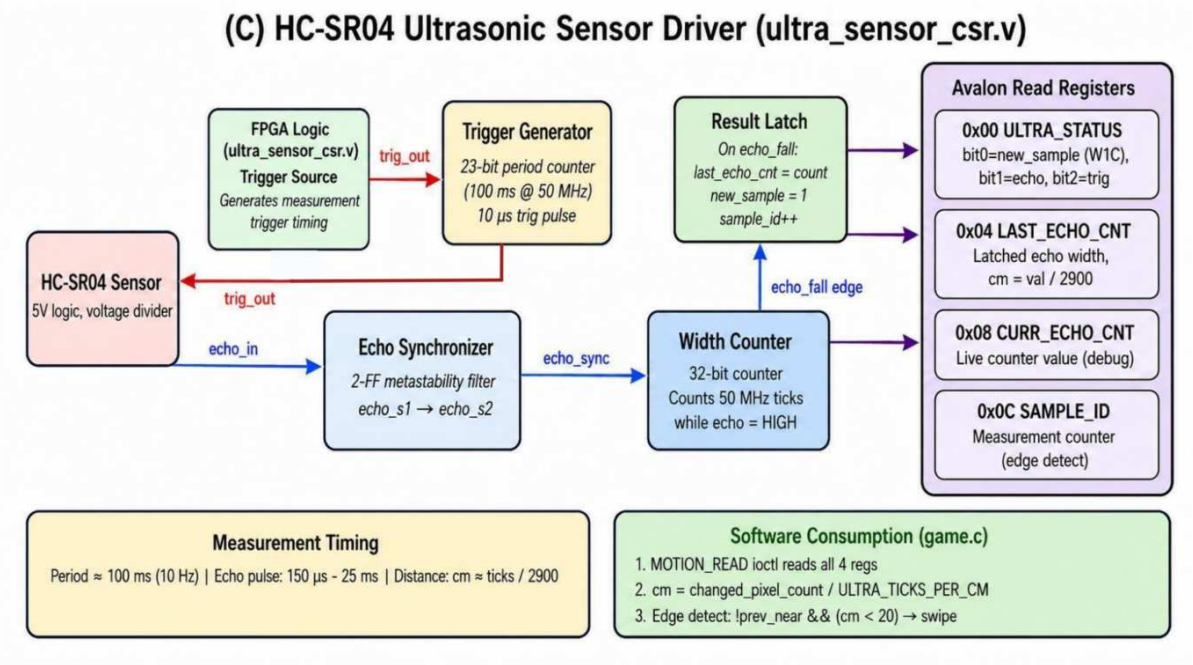

Figure 3: Ultrasonic sensor CSR structure, including trigger generation, echo synchronization, echo counting, sample latching, and register readback.

clock ticks. On the falling edge of echo, the module latches the counter into `last_echo_cnt`, increments `sample_id`, and sets `new_sample`. Software clears `new_sample` by writing 1 to bit 0 of the status register.

The approximate distance conversion used by the software and test tools is:

$$\text{distance in cm} \approx \frac{\text{echo ticks}}{2900}.$$

This follows from a 50 MHz counter and the HC-SR04 round-trip sound propagation relation.

3.3 VGA Timing and Pixel Composition

The VGA timing generator is `rtl/vga_counter.sv`. It receives the 50 MHz clock and advances horizontal and vertical counters for 640 by 480 active video. The pixel column is derived from `hcount[10:1]`, so each visible pixel lasts two 50 MHz clock cycles and `VGA_CLK` is generated from `hcount[0]`.

The renderer in `top_module.sv` is a zero-framebuffer design. It does not store a full 640 by 480 image. Instead, each output pixel is selected from several sources:

1. a repeating 128 by 96 background pattern formed from twelve 32 by 32 RGB888 tile ROMs;
2. up to three visible object slots, each using a rotated 64 by 64 RGBA4444 sprite;
3. a dashed horizontal cut line at $y = 320$, with a 16-pixel on/off pattern;
4. a top-left numeric score;
5. a top-center `LEVEL` display derived from score;
6. top-right heart icons derived from the lives register;

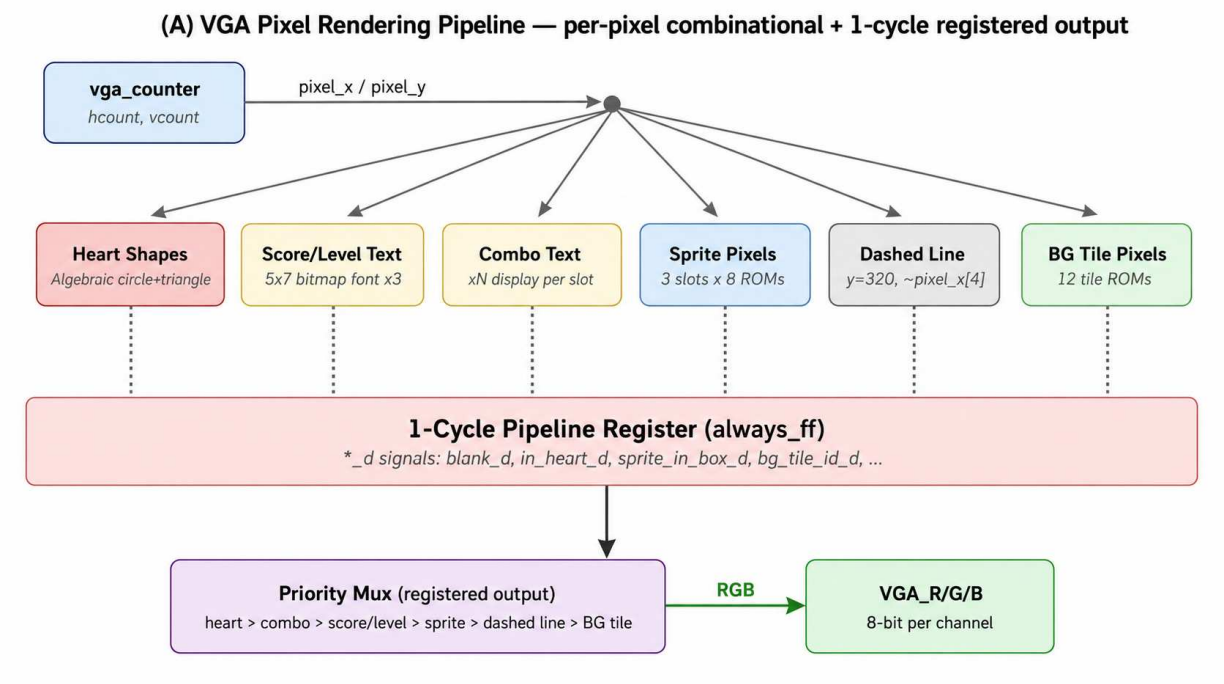

Figure 4: VGA rendering pipeline, including timing generation, background tile lookup, sprite lookup, HUD logic, and final pixel priority selection.

7. optional yellow combo text near a cut fruit.

Because `sprite_rom` and `tile_rom_rgb888` are synchronous ROMs, the RTL delays non-ROM classification signals by one clock cycle to align them with ROM pixel data. The final pixel priority in the output mux is:

$$\text{hearts} > \text{combo text} > \text{score/level text} > \text{sprite} > \text{dashed line} > \text{background}.$$

3.4 Sprite ROMs and Rotation

The module `rtl/sprite_rom.sv` implements a 4096 by 16-bit ROM initialized by `$readmemh`. Each sprite is 64 by 64 pixels in RGBA4444 format. The renderer instantiates eight sprite ROMs for each of three object slots: apple, apple blown, banana, banana blown, watermelon, watermelon blown, bomb, and boom blown. The alpha nibble is used as a binary transparency test; nonzero alpha produces an opaque sprite pixel.

Rotation is implemented by inverse mapping from screen pixel coordinates back into sprite texture coordinates. For each object slot, the RTL computes the current pixel's signed displacement from the sprite center, rejects pixels outside a conservative 46-pixel bounding box, looks up sine and cosine from `rtl/rot_lut.sv`, applies a fixed-point inverse rotation, and addresses the sprite ROM if the resulting source coordinate is inside the 64 by 64 sprite.

3.5 Graphics Assets and ROM Initialization

The repository stores graphics under `pics/`. These files are not PNG or JPG files; they are synthesis-time ROM initialization files used by `$readmemh`. The sprite files contain 4096 16-bit RGBA4444 words,

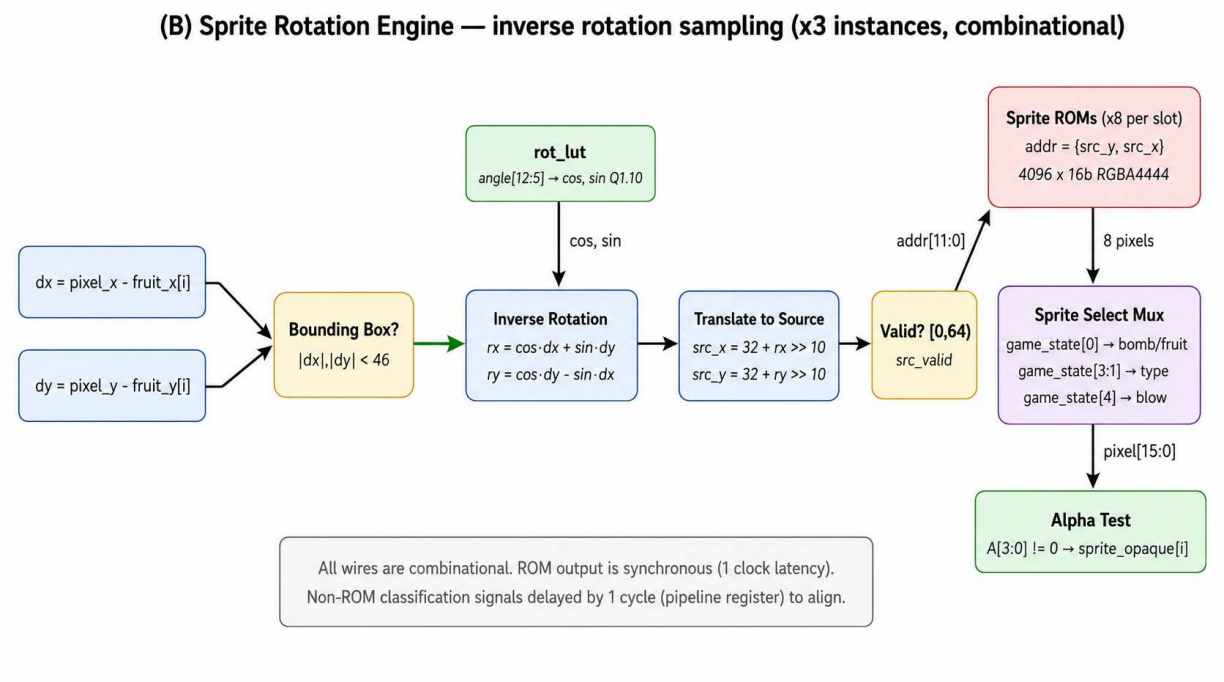

Figure 5: Sprite rendering path for one object slot, including inverse rotation, source-coordinate validation, ROM address generation, texture selection, and alpha test.

one word per pixel. The background tile files contain 1024 24-bit RGB888 words, one word per pixel. Figures 6 and 7 are generated previews of the same data used by the RTL.

The renderer duplicates the eight sprite ROMs once per object slot. This is resource-expensive, but it gives each slot an independent ROM read address in the same cycle. The background uses twelve tile ROMs with a common local tile address; the selected tile output is chosen after a one-cycle delay so it aligns with the synchronous ROM outputs.

3.6 Display and HUD Timing

The display pipeline does not wait for a complete frame of game state. Registers written by the HPS are sampled directly by the pixel logic, and the output changes as soon as raster timing reaches affected pixels. The score and level display are derived from `score_reg`; the level threshold in RTL matches the software thresholds of 10 and 20 points. Lives are represented by three top-right heart shapes, with red hearts for remaining lives and gray hearts for lost lives. Combo text is only visible for combo counts of two or greater and is clipped to remain on screen.

4 Software Design

4.1 Kernel Driver

The Linux kernel module `software/top_module.c` implements a platform driver and misc character device named `/dev/top_module`. The driver binds to the device-tree compatible string `csee4840,top_module-1.0`. During probe, it obtains the device-tree memory resource, requests the memory region, maps it with `of_iomap()`, and registers the misc device.

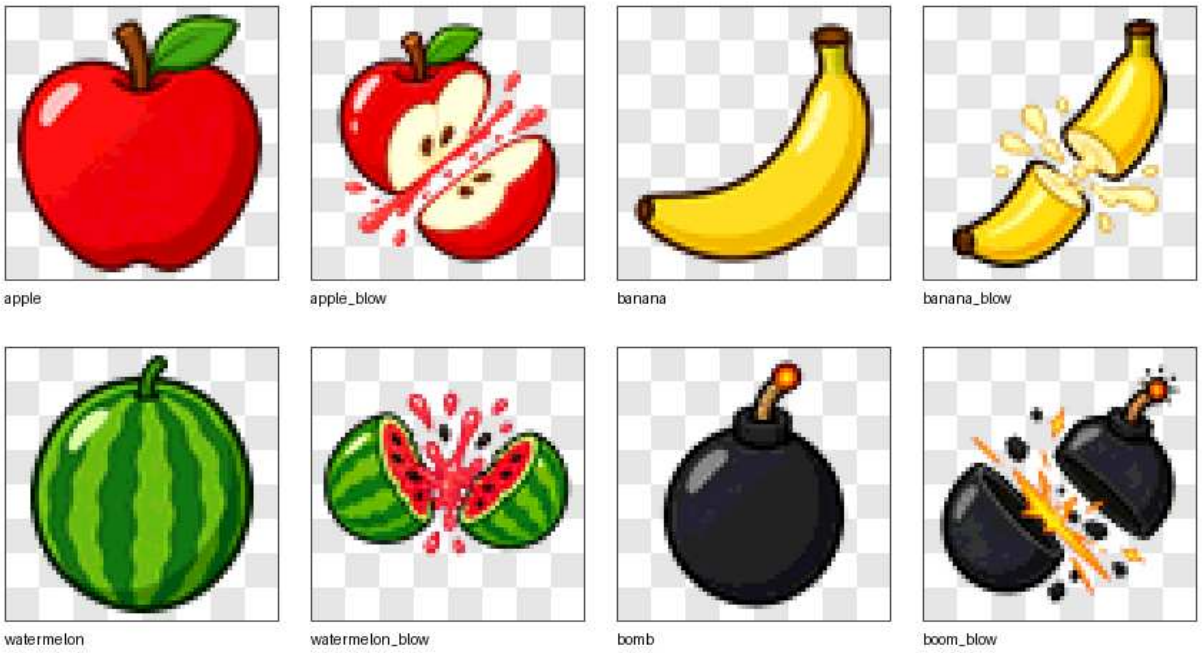

Figure 6: Preview generated from the eight 64 by 64 RGBA4444 sprite ROM initialization files in `pics/`.

The driver exposes five ioctl commands defined in `software/top_module.h`. These commands hide raw MMIO details from the user-space game. Register access is implemented with `ioread32()` and `iowrite32()` at byte offsets within the mapped 64-byte register window.

Table 3: Driver ioctl commands

Command	Direction	Behavior
`MOTION_READ`	FPGA to user	Reads ultrasonic status, last echo count, and sample ID into a three-word `motion_status_t`; clears `new_sample` with W1C if set
`FRUIT_WRITE`	user to FPGA	Writes selected slot, x, y, radius, and visible flag
`SCORE_WRITE`	user to FPGA	Writes the global score register
`GAME_ST_WRITE`	user to FPGA	Writes packed object render state; the driver also writes `OBJECT_INDEX` from bits [17:16]
`LIVES_WRITE`	user to FPGA	Writes lives masked to two bits

4.2 User-Space Game Loop

The game executable is implemented in `software/game.c`. It opens `/dev/top_module`, initializes score and lives, spawns the first wave, and then runs until SIGINT or until lives reach zero. The loop is paced with `usleep(FRAME_US)`, where `FRAME_US = 33000`, so the intended update rate is roughly 30 frames per second.

The main software state is an array of three `object_t` structures. Each object stores position, velocity, start position, age in frames, radius, visibility, object type, fruit sprite ID, blown state, combo display state, cut state, and rotation angle. Positions use a small fixed-point scale factor `Q = 4` for vertical position and velocity.

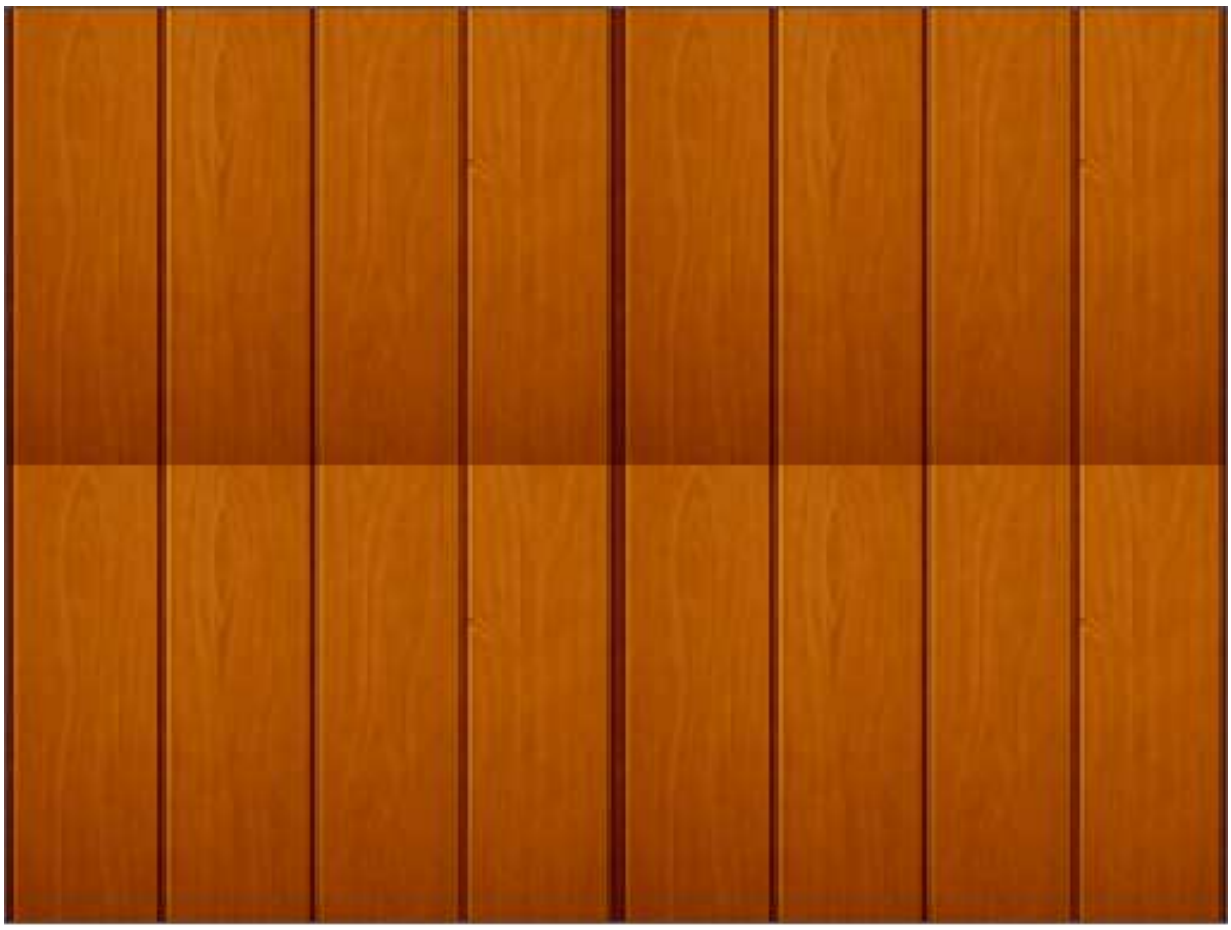

Figure 7: Preview generated from the twelve 32 by 32 RGB888 background tile ROM initialization files in `pics/tiles_32_rgb888_hex/`.

The combo healing rule is implemented in the main collision loop. After a fruit cut increments `combo_count`, the code checks `combo_count % COMBO_HEAL_STEP`; with `COMBO_HEAL_STEP = 10`, every tenth consecutive fruit combo restores one life when the player has fewer than `MAX_LIVES = 3`.

Table 4: Major user-space functions in `software/game.c`

Function	Implemented responsibility
`level_for_score()` and `refresh_level()`	Convert score thresholds into the current level and print a level-up message when the value changes
`write_fruit_hw()`	Clamp object coordinates to the 10-bit hardware register range and send slot, position, radius, and visibility through `FRUIT_WRITE`
`write_state_hw()`	Pack bomb flag, fruit ID, blow bit, angle, slot ID, combo count, and combo offsets into `GAME_STATE`
`spawn_object()`	Choose spawn zone, radius, object type, sprite ID, rotation speed, horizontal velocity, and vertical speed for one slot
`spawn_wave()` and `wave_done()`	Activate one, two, or three object slots depending on level and decide when a new wave should begin
`update_object()`	Advance position and rotation, handle blow-sprite timeout, bounce at horizontal boundaries, and handle missed objects
`check_swipe()`	Read ultrasonic status, detect new samples, convert ticks to centimeters, and emit only a far-to-near edge
`object_is_cuttable()`	Test whether an object is visible, uncut, and close enough to the horizontal cut line
`main()`	Initialize the device, run the frame loop, process collisions, update score/lives/combo, and clear objects at exit

4.3 Object Spawning, Motion, and Rules

The function `spawn_object()` chooses one of three spawn zones: top-center, upper-left, or upper-right. It randomly selects a radius between 22 and 34 pixels, assigns fruit or bomb using `BOMB_PERCENT = 30`, chooses one of three fruit sprites for fruit objects, assigns one of three rotation increments, and computes a vertical velocity so each slot reaches the cut line at a staggered target frame. Slot 0 targets about frame 24, slot 1 about frame 38, and slot 2 about frame 52, with a jitter of plus or minus three frames.

The function `update_object()` advances non-blown objects by one frame, updates rotation, recomputes vertical position from starting position and velocity, applies horizontal drift, and bounces at the left and right screen edges. If an uncut fruit exits below `EXIT_Y = SCREEN_H + 64`, software decrements lives and resets combo. Bombs that exit below the screen are logged as dodged and do not penalize the player.

The function `object_is_cuttable()` checks whether an object is visible, not already cut, and within radius + 24 pixels of the horizontal cut line. The extra 24 pixels are an implemented gameplay tolerance.

4.4 Swipe Detection

The function `check_swipe()` reads `MOTION_READ`. It treats a sample as new if either the returned `new_sample` flag is set or the returned sample ID differs from the last seen sample ID. It converts the last

echo count to centimeters using `ULTRA_TICKS_PER_CM = 2900` and then detects only the far-to-near transition:

$$\text{swipe} = \neg\text{prev\_near} \wedge \text{curr\_near},$$

where curr_near is true when the measured distance is greater than zero and less than 20 cm.

The software does not continuously trigger swipes while a hand remains near the sensor. The `prev_near` state prevents repeated cuts until the measured distance first returns to far and then becomes near again. This is important because the HC-SR04 sample period is much slower than the VGA refresh rate; without edge detection, one held hand position could be counted repeatedly across multiple game frames.

5 Hardware-Software Interface

The only regular control path between HPS software and custom FPGA logic is memory-mapped I/O through the lightweight HPS-to-FPGA bridge. The Platform Designer system maps the custom peripheral at `0xFF200000` with a 64-byte span. The generated device tree contains a `top_module_0` node with `reg = <0x00000001 0x00000000 0x00000040>`, which the Linux driver maps through the platform-device resource.

The register map is word-addressed on the Avalon side but byte-addressed in the Linux driver header. For example, Avalon word address 4 corresponds to byte offset `0x10`. The driver performs byte-offset arithmetic on the kernel virtual base returned by `of_iomap()`, while `top_module.sv` receives the low four word-address bits from Platform Designer.

Table 5: Avalon-MM register map

Byte offset	Word	Name	Access	Description
0x00	0	ULTRA_STATUS	R/W1C	bit0 = new sample, bit1 = synchronized echo, bit2 = trigger; writing bit0 clears new sample
0x04	1	LAST_ECHO_CNT	R	Latched echo high-time in 50 MHz clock ticks
0x08	2	CURR_ECHO_CNT	R	Current echo counter, mainly for debug
0x0C	3	SAMPLE_ID	R	Increments after each completed echo measurement
0x10	4	FRUIT_X	R/W	X coordinate for selected object slot
0x14	5	FRUIT_Y	R/W	Y coordinate for selected object slot
0x18	6	FRUIT_RADIUS	R/W	Radius/collision parameter for selected object slot
0x1C	7	FRUIT_CTRL	R/W	bit0 = selected object visible
0x20	8	SCORE	R/W	Global score; RTL also derives displayed level from this value
0x24	9	GAME_STATE	R/W	Packed per-slot sprite type, blow state, angle, slot, and combo display data
0x28	10	LIVES	R/W	Lives value, 0 to 3, rendered as hearts
0x2C	11	OBJECT_INDEX	R/W	Selected object slot for coordinate/radius/visibility register writes

5.1 Register Update Order

For position-related updates, the driver first writes `OBJECT_INDEX`, then writes `FRUIT_X`, `FRUIT_Y`, `FRUIT_RADIUS`, and `FRUIT_CTRL`. The selected slot is therefore held in the FPGA register file while the following coordinate/control writes arrive.

For packed game-state updates, `game.c` embeds the target slot in bits [17:16] of `GAME_STATE`. The driver writes `OBJECT_INDEX` from those bits for consistency and then writes the packed state word. The RTL uses the embedded slot when selecting which `game_state_reg` entry to update, falling back to `object_index_safe` only if the embedded slot is outside the implemented 0–2 range.

5.2 Physical Pin and Board Connections

The board wrapper `DE1_SoC_GHRD/de1_soc_top.v` connects `GPIO_0[0]` to the sensor echo input and drives `GPIO_1[0]` from the sensor trigger output. The Quartus pin assignment files map those GPIO pins to package pins and specify 3.3-V LVTTL I/O. The HC-SR04 echo output is a 5-V signal, so the external voltage divider described in the repository README is used before connecting to the FPGA input.

Table 6: `GAME_STATE` bit fields

Bits	Field	Meaning
[0]	is_bomb	0 selects fruit sprite family; 1 selects bomb or explosion sprite family
[3:1]	fruit_id	Fruit sprite ID: 0 apple, 1 banana, 2 watermelon; other values fall through to apple in RTL
[4]	blow	0 normal sprite, 1 cut/blown sprite variant
[12:5]	angle	8-bit rotation angle, where 0 to 255 represents one full turn
[15:13]	reserved	Not used by current RTL
[17:16]	slot	Object slot selector embedded in the state word; RTL uses it when writing `game_state_reg`
[23:18]	combo	Combo count to display; values below 2 hide combo text
[27:24]	combo_dx	Combo x-offset index, mapped by RTL to several positive or negative pixel offsets
[31:28]	combo_dy	Combo y-offset index, using the same offset mapping as `combo_dx`

6 Implementation Details

6.1 Repository Structure

The project source tree is organized around the hardware/software boundary. The `rtl/` directory contains the custom FPGA logic, `software/` contains the Linux driver interface and the game program, `pics/` contains ROM initialization data for graphics, `tools/` contains sensor diagnostics, and `DE1_SoC_GHRD/` contains the Quartus and Platform Designer integration files used to connect the custom peripheral to the DE1-SoC base system.

```
fruit_ninja/
  software/
    Makefile
    game.c
    top_module.c
    top_module.h
  rtl/
    top_module.sv
    ultra_sensor_csr.v
    vga_counter.sv
    rot_lut.sv
    sprite_rom.sv
    tile_rom_rgb888.sv
  pics/
```

```
    apple.hex
    apple_blow.hex
    banana.hex
    banana_blow.hex
    bomb.hex
    boom_blow.hex
    watermelon.hex
    watermelon_blow.hex
    tiles_32_rgb888_hex/
      0000.hex ... 0011.hex
  tools/
    ultra_read.c
    ultra_sonic.sh
  DE1_SoC_GHRD/
    DE1_SoC_GHRD.qpf
    DE1_SoC_GHRD.qsf
    DE1_SoC_pin_assignments.qsf
    de1_soc_top.v
    top_module_hw.tcl
    soc_system.qsys
    soc_system.dts
    PLL/
```

Listing 1: Project source tree used for the implementation.

6.2 Build and Run Flow

The repository's software build flow is defined in `software/Makefile`. By default, `make` builds both the kernel module and the game executable. The Makefile defaults to an ARM hard-float cross compiler and to kernel headers under the running system's `/usr/src` header directory.

The intended software run sequence is:

1. build the FPGA design from the Quartus project under `DE1_SoC_GHRD`;
2. boot Linux with a device tree containing the `top_module` node;
3. run `cd software && make` on the target or cross-build environment;
4. load the driver with `insmod top_module.ko`;
5. run the game with `./game`;
6. unload with `rmmod top_module` after exiting.

The direct diagnostic tools bypass the kernel module and read the ultrasonic registers through physical address `0xFF200000`. They are useful for checking sensor timing before running the full game.

File and module responsibilities

Path	Responsibility
rtl/top_module.sv	Custom FPGA peripheral top level: Avalon decoder/register file, VGA renderer, sprite/tile ROM instances, HUD, LEDs, ultrasonic CSR instantiation
rtl/ultra_sensor_csr.v	HC-SR04 trigger generation, echo timing, sample registers, W1C status flag
rtl/vga_counter.sv	VGA timing counters, sync, blanking, and 25 MHz VGA clock generation
rtl/rot_lut.sv	8-bit angle to signed Q1.10 sine/cosine lookup
rtl/sprite_rom.sv	64 by 64 RGBA4444 sprite ROM wrapper
rtl/tile_rom_rgb888.sv	32 by 32 RGB888 tile ROM wrapper
pics/*.hex	Fruit, bomb, and cut/explosion sprite data initialized into ROMs at synthesis time
pics/tiles_32_rgb888_hex/*.hex	Twelve background tile images initialized into RGB888 tile ROMs
software/top_module.h	Register offsets, ioctl numbers, shared structs, and game constants
software/top_module.c	Linux platform/misc driver for the custom peripheral
software/game.c	User-space game loop, physics, rules, sensor polling, and register writes
software/Makefile	Builds the kernel module and user-space executable
tools/ultra_read.c	Direct /dev/mem ultrasonic register diagnostic tool
tools/ultra_sonic.sh	Shell-based ultrasonic register diagnostic using devmem
tools/clean.sh	Removes common build artifacts from the project tree
DE1_SoC_GHRD/de1_soc_top.v	Board-level wrapper that connects VGA, LEDs, HPS, and ultrasonic GPIO signals
DE1_SoC_GHRD/top_module_hw.tcl	Platform Designer component definition for the custom Avalon peripheral and graphics ROM files
DE1_SoC_GHRD/soc_system.qsys	Platform Designer system connection, including the HPS lightweight bridge path to top_module_0
DE1_SoC_GHRD/soc_system.dts	Device-tree description that exposes the custom peripheral to Linux
DE1_SoC_GHRD/DE1_SoC_pin_assignments.qsf	Board pin assignments for GPIO, VGA, LEDs, clocks, and other DE1-SoC signals

7 Testing and Evaluation

The project was checked through ultrasonic register diagnostics, driver-level software integration, VGA rendering inspection, and end-to-end gameplay observation. Table 7 summarizes the validation coverage.

7.1 Resource Utilization and Performance

The final Quartus fitter and TimeQuest timing reports are the authoritative source for ALM/LE count, exact M10K packing, inferred DSP use, I/O count, and Fmax. The values below are source-derived estimates and timing values that can be read directly from the RTL and C constants; the fitted values for the submitted bitstream may differ after synthesis, packing, and timing analysis.

The main FPGA resource pressure is block RAM. The RTL instantiates eight 64 by 64 RGBA4444 sprite ROMs for each of three object slots, for 24 sprite ROM instances total. Each sprite ROM stores

Table 7: Validation coverage

Test	Procedure	Observed result
Ultrasonic input path	Run the game with the HC-SR04 connected and move a hand through the near-field threshold used by `software/game.c`	Near hand movement drives the swipe input used by the game loop
Driver probe and ioctl path	Load `top_module.ko`, open `/dev/top_module`, and run the user-space game	The integrated demo exercises the ioctl path used to read sensor state and update FPGA display registers
VGA rendering	Program the FPGA and inspect display for tiled background, cut line, score, level text, combo text, hearts, and sprites	Figure 8 shows the live game interface rendered on the VGA display
Gameplay rules	Cut fruit, cut bombs, miss fruit, and let bombs fall off-screen; compare score, life, combo, and sprite-state behavior to `software/game.c`	Score, combo display, heart count, and sliced-fruit sprite changes were observed during the demo
Level progression	Reach the score thresholds implemented in `software/game.c`	The displayed level changes as the score increases

$4096 \times 16 = 65{,}536$ bits, which maps to about seven 10-Kbit M10K blocks if implemented as a separate ROM. The background path instantiates twelve 32 by 32 RGB888 tile ROMs. Each tile ROM stores $1024 \times 24 = 24{,}576$ bits, or about three M10K blocks per tile. Using the DE1-SoC Cyclone V device capacity of 397 M10K blocks, this source-level estimate is approximately 204 M10K blocks, or about 51% of the available M10K memory.

Table 8: Implementation-derived FPGA resource estimates

Resource item	Source-derived quantity	Basis
Sprite ROM memory	24 ROMs; about 168 M10K	Eight sprite ROMs are duplicated for each of the three object slots; each ROM stores 65,536 bits
Background tile memory	12 ROMs; about 36 M10K	Twelve RGB888 tile ROMs provide the tiled background; each ROM stores 24,576 bits
Sprite and background ROM total	about 204 of 397 M10K, or about 51%	Estimate assumes separate ROM packing into M10K blocks
Project-specific external I/O	41 signals	29 VGA signals, 10 LED signals, and two ultrasonic sensor GPIO signals at the custom peripheral boundary
ALMs/LEs and DSP blocks	synthesis-dependent	Exact logic packing and DSP inference require the Quartus fitter report

8 Results and Discussion

The repository supports the following implementation-level results:

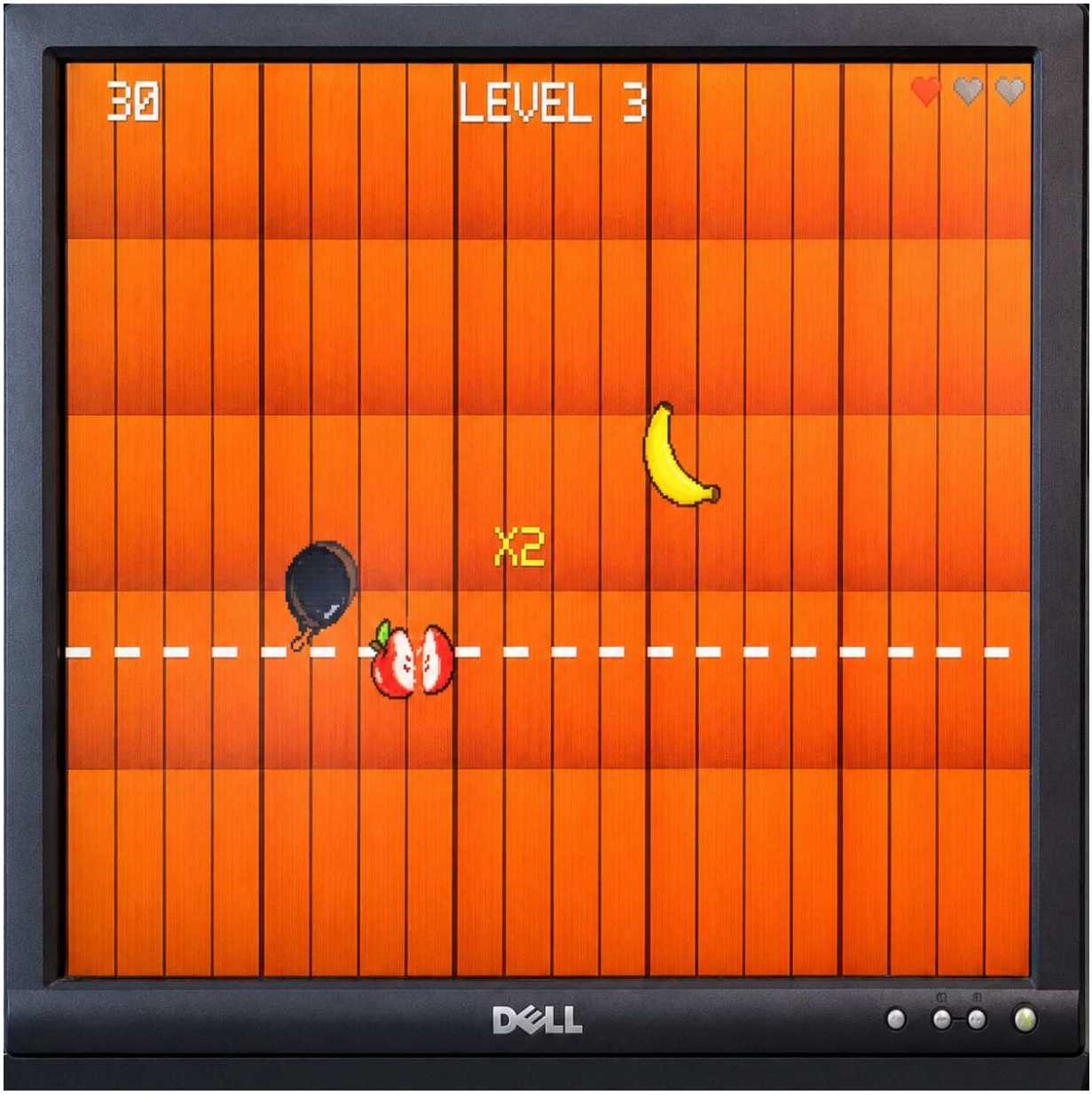

Figure 8: Game demonstration screenshot showing VGA output with background, cut line, score, level, hearts, combo text, and active sprites.

Table 9: Source-derived timing and performance values

Item	Source-derived value and interpretation
System clock	50 MHz main clock for the Avalon register file, ultrasonic CSR, and VGA timing counter
VGA raster rate	50 MHz/(1600 × 525) ≈ 59.5 Hz, derived from `vga_counter.sv`
User-space game loop	`FRAME_US = 33000`, giving an intended update rate of approximately 30.3 frames/s before Linux scheduling overhead
Ultrasonic measurement period	approximately 100 ms from the `period_cnt == 5,000,000` reset condition at 50 MHz
Trigger pulse width	4999 cycles, approximately 99.98 μs, from `period_cnt > 100` and `period_cnt < 5100`
Input response bound from source timing	about 150 ms plus software scheduling and sensor analog effects: one ultrasonic period, one software frame interval, and up to one display scan dominate the visible response path
Fmax and timing slack	compilation-dependent; the final value must come from TimeQuest timing analysis for the Quartus build

- The HPS-to-FPGA register interface is defined consistently across `top_module.sv`, `top_module.h`, and `top_module.c`.
- The ultrasonic CSR provides a software-readable echo count, current count, sample ID, and W1C new-sample flag.
- The user-space game loop implements fruit, bomb, combo, lives, level progression, and blow-sprite display rules.
- The VGA renderer implements a zero-framebuffer pipeline using tile ROMs, sprite ROMs, fixed-point rotation, alpha testing, and HUD priority composition.

Figure 8 shows the integrated game display. The demonstrated interface renders the background, object sprites, cut line, score, level, combo text, and heart icons together. During gameplay, the heart count changes according to hit, miss, bomb, and combo-heal events; the combo display updates; sliced fruit switches to the alternate sprite state; the score increments after fruit cuts; and the level indicator advances as the score reaches the thresholds implemented in software.

The code also shows several design tradeoffs. Moving game policy to C makes threshold and gameplay tuning easy, while keeping VGA generation in RTL avoids moving pixel data over the HPS-FPGA bridge. The main input limitation is the implemented ultrasonic measurement period of about 100 ms. Since the game loop targets about 33 ms per frame, several game frames may pass between new sensor samples.

9 Limitations and Future Work

- **Sensor update rate:** The ultrasonic CSR currently triggers about every 100 ms. Reducing the period counter threshold could lower input latency, subject to HC-SR04 timing limits and empirical validation.
- **Threshold calibration:** `NEAR_CM = 20` is a compile-time constant. A runtime calibration mode or ioctl-configurable threshold would make the game easier to tune for different setups.
- **Legacy naming:** Some ioctl and struct field names are inherited from a previous motion-detection concept. In the current implementation, those fields mean ultrasonic new-sample flag, echo ticks, and sample ID.

- **Physics model:** Vertical motion is linear in the implemented game. A future version could use acceleration or a parabolic trajectory if more realistic fruit motion is desired.
- **Audio feedback:** The current implementation does not include sound effects.

10 Conclusion

The implemented project is a hardware/software co-design of an ultrasonic Fruit Ninja-style game on the DE1-SoC. The HPS side manages flexible game behavior in C through a Linux driver and ioctl interface. The FPGA side implements the time-sensitive ultrasonic counter and all VGA pixel generation, including sprite rotation and HUD rendering, without a frame buffer. The design demonstrates a clear division of labor: software handles decisions and state transitions, while hardware handles deterministic I/O, register storage, ROM lookup, and per-pixel display composition.

A Appendix: Complete Project Source Listings

This appendix contains every source file under `software/` and `rtl/` in full. The `DE1_SoC_GHRD` directory is much larger and contains generated or vendor-style project support files, so only the project-specific integration excerpts that explain how the custom peripheral is connected are included.

A.1 Complete Software Directory

```
# Makefile for Fruit-Ninja DE1-SoC software
#
# Two targets:
#   make game     - build userspace game program (cross-compile or native)
#   make module   - build Linux kernel module  (needs kernel headers)

# Cross-compiler prefix (leave empty when compiling natively on DE1-SoC)
CROSS_COMPILE ?= arm-linux-gnueabihf-
CC            := $(CROSS_COMPILE)gcc

ifneq (${KERNELRELEASE},)

# Called from kernel build system
obj-m := top_module.o

else

KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
PWD := $(shell pwd)

default: module game

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

# ---------- Userspace game ----------
game: game.c top_module.h
    $(CC) -O2 -Wall -o $@ game.c

# ---------- Utilities ----------
clean:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
    ${RM} game

TARFILES = Makefile README top_module.h top_module.c game.c
TARFILE = fruit_ninja_sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
    tar zcf $(TARFILE) $(TARFILES)

endif
```

Listing 2: `software/Makefile`: software build targets for the kernel module and user-space game.

```
#ifndef _TOP_MODULE_H
#define _TOP_MODULE_H

#ifdef __KERNEL__
#include <linux/ioctl.h>
#else
#include <sys/ioctl.h>
#endif

/* ---- Hardware register byte offsets (Avalon-MM, 32-bit) ----
```

```
 *
 * Ultrasonic section (read):
 *   0x00 ULTRA_STATUS    bit0=new, bit1=echo, bit2=trig.
 *                        Writing 1 to bit0 clears the "new" flag.
 *   0x04 LAST_ECHO_CNT   50-MHz ticks of last echo pulse.  cm ≈ val / 2900.
 *   0x08 CURR_ECHO_CNT   in-progress counter (debug only).
 *   0x0C SAMPLE_ID       increments once per completed measurement.
 *
 * Game section (write-only from HPS):
 *   0x10 FRUIT_X         0..639
 *   0x14 FRUIT_Y         0..479
 *   0x18 FRUIT_RADIUS    px
 *   0x1C FRUIT_CTRL      bit0 = visible
 *   0x20 SCORE
 *   0x24 GAME_STATE       bit0=bomb, bits[3:1]=fruit sprite id,
 *                         bit4=blow sprite, bits[12:5]=rotation angle,
 *                         bits[17:16]=slot, bits[23:18]=combo count,
 *                         bits[27:24]/[31:28]=combo x/y offset indices
 *   0x28 LIVES            0..3, rendered as top-right hearts
 *   0x2C OBJECT_INDEX     selected object slot for FRUIT_X/GAME_STATE writes
 */
#define REG_ULTRA_STATUS     0x00
#define REG_LAST_ECHO_CNT    0x04
#define REG_CURR_ECHO_CNT    0x08
#define REG_SAMPLE_ID        0x0C
#define REG_FRUIT_X          0x10
#define REG_FRUIT_Y          0x14
#define REG_FRUIT_RADIUS     0x18
#define REG_FRUIT_CTRL       0x1C
#define REG_SCORE            0x20
#define REG_GAME_STATE       0x24
#define REG_LIVES            0x28
#define REG_OBJECT_INDEX     0x2C

#define ULTRA_STATUS_NEW     0x1
#define ULTRA_STATUS_ECHO    0x2
#define ULTRA_STATUS_TRIG    0x4

/* 50 MHz counter, sound 343 m/s round-trip => ticks/cm ≈ 2915 */
#define ULTRA_TICKS_PER_CM   2900u

/* ---- Data structures shared between kernel and user space ---- */
/* Ultrasonic sample snapshot returned by MOTION_READ (name kept for
 * compatibility with the legacy driver architecture). */
struct motion_status {
    unsigned int motion_detected;   /* 1 if new sample arrived this read */
    unsigned int changed_pixel_count; /* last_echo_cnt (ticks) */
    unsigned int frame_counter;     /* sample_id */
};
typedef struct motion_status motion_status_t;

struct fruit_params {
    unsigned int slot;
    unsigned int x;
    unsigned int y;
    unsigned int radius;
    unsigned int visible;
};
typedef struct fruit_params fruit_params_t;

/* ---- ioctl definitions (same magic/numbers as legacy driver) ---- */
#define FRUTNINJA_MAGIC  'F'
#define MOTION_READ    _IOC(_IOC_READ,  FRUTNINJA_MAGIC, 1, 12)
#define FRUIT_WRITE    _IOC(_IOC_WRITE, FRUTNINJA_MAGIC, 2, 20)
#define SCORE_WRITE    _IOC(_IOC_WRITE, FRUTNINJA_MAGIC, 3, 4)
#define GAME_ST_WRITE  _IOC(_IOC_WRITE, FRUTNINJA_MAGIC, 4, 4)
#define LIVES_WRITE    _IOC(_IOC_WRITE, FRUTNINJA_MAGIC, 5, 4)

```

```
/* ---- Game constants ---- */
#define SCREEN_W    640
#define SCREEN_H    480

#endif /* _TOP_MODULE_H */
```

Listing 3: `software/top_module.h`: complete register, struct, ioctl, and screen-constant definitions.

```
/*
 * top_module.c  --  Linux platform/misc driver for Fruit-Ninja FPGA peripheral
 *                   (ultrasonic version; structure copied from legacy camera driver)
 *
 * Exposes /dev/top_module with ioctl interface for:
 *   MOTION_READ     - snapshot (new_flag, last_echo_cnt, sample_id); clears "new"
 *   FRUIT_WRITE     - push fruit slot (x, y, radius, visible) to FPGA
 *   SCORE_WRITE     - write score register
 *   GAME_ST_WRITE   - write selected slot game_state register
 *   LIVES_WRITE     - write lives register for heart display
 */

#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 "top_module.h"

#define DRIVER_NAME "top_module"

/* ---- device state ---- */
struct fruit_ninja_dev {
    struct resource res;
    void __iomem   *virtbase;
} dev;

/* ---- low-level register helpers ---- */
static inline u32 reg_read(unsigned int offset)
{
    return ioread32((u8 *)dev.virtbase + offset);
}

static inline void reg_write(unsigned int offset, u32 value)
{
    iowrite32(value, (u8 *)dev.virtbase + offset);
}

/* ---- ioctl ---- */
static long top_module_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    unsigned int    ms[3];
    unsigned int    fp[5];
    unsigned int    val, status;

    switch (cmd) {

    case MOTION_READ:
        status = reg_read(REG_ULTRA_STATUS);
        ms[0] = status & ULTRA_STATUS_NEW;
        ms[1] = reg_read(REG_LAST_ECHO_CNT);
        ms[2] = reg_read(REG_SAMPLE_ID);
```

```
        /* write 1 to bit0 clears the new_sample flag */
        if (ms[0])
            reg_write(REG_ULTRA_STATUS, ULTRA_STATUS_NEW);
        if (copy_to_user((void __user *)arg, &ms, sizeof(ms)))
            return -EACCES;
        break;

    case FRUIT_WRITE:
        if (copy_from_user(&fp, (void __user *)arg, sizeof(fp)))
            return -EACCES;
        reg_write(REG_OBJECT_INDEX, fp[0] % 3);
        reg_write(REG_FRUIT_X,      fp[1] & 0x3FF);
        reg_write(REG_FRUIT_Y,      fp[2] & 0x3FF);
        reg_write(REG_FRUIT_RADIUS, fp[3] & 0x3FF);
        reg_write(REG_FRUIT_CTRL,   fp[4] & 1);
        break;

    case SCORE_WRITE:
        if (copy_from_user(&val, (void __user *)arg, sizeof(val)))
            return -EACCES;
        reg_write(REG_SCORE, val);
        break;

    case GAME_ST_WRITE:
        if (copy_from_user(&val, (void __user *)arg, sizeof(val)))
            return -EACCES;
        reg_write(REG_OBJECT_INDEX, (val >> 16) & 3);
        reg_write(REG_GAME_STATE, val);
        break;

    case LIVES_WRITE:
        if (copy_from_user(&val, (void __user *)arg, sizeof(val)))
            return -EACCES;
        reg_write(REG_LIVES, val & 3);
        break;

    default:
        return -EINVAL;
    }
    return 0;
}

/* ---- file ops ---- */
static const struct file_operations top_module_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = top_module_ioctl,
};

static struct miscdevice top_module_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name  = DRIVER_NAME,
    .fops  = &top_module_fops,
};

/* ---- probe / remove ---- */
static int __init top_module_probe(struct platform_device *pdev)
{
    int ret;

    ret = misc_register(&top_module_misc_device);
    if (ret)
        return ret;

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

```
129
130     if (!request_mem_region(dev.res.start, resource_size(&dev.res),
131                 DRIVER_NAME)) {
132         ret = -EBUSY;
133         goto out_deregister;
134     }
135
136     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
137     if (!dev.virtbase) {
138         ret = -ENOMEM;
139         goto out_release;
140     }
141
142     pr_info(DRIVER_NAME ": mapped to %p, size %u\n",
143         dev.virtbase, (unsigned)resource_size(&dev.res));
144     return 0;
145
146 out_release:
147     release_mem_region(dev.res.start, resource_size(&dev.res));
148 out_deregister:
149     misc_deregister(&top_module_misc_device);
150     return ret;
151 }
152
153 static int top_module_remove(struct platform_device *pdev)
154 {
155     iounmap(dev.virtbase);
156     release_mem_region(dev.res.start, resource_size(&dev.res));
157     misc_deregister(&top_module_misc_device);
158     return 0;
159 }
160
161 /* ---- device-tree match ---- */
162 #ifdef CONFIG_OF
163 static const struct of_device_id top_module_of_match[] = {
164     { .compatible = "csee4840,top_module-1.0" },
165     {},
166 };
167 MODULE_DEVICE_TABLE(of, top_module_of_match);
168 #endif
169
170 static struct platform_driver top_module_driver = {
171     .driver = {
172         .name           = DRIVER_NAME,
173         .owner          = THIS_MODULE,
174         .of_match_table = of_match_ptr(top_module_of_match),
175     },
176     .remove = __exit_p(top_module_remove),
177 };
178
179 static int __init top_module_init(void)
180 {
181     pr_info(DRIVER_NAME ": init\n");
182     return platform_driver_probe(&top_module_driver, top_module_probe);
183 }
184
185 static void __exit top_module_exit(void)
186 {
187     platform_driver_unregister(&top_module_driver);
188     pr_info(DRIVER_NAME ": exit\n");
189 }
190
191 module_init(top_module_init);
192 module_exit(top_module_exit);
193
194 MODULE_LICENSE("GPL");
195 MODULE_AUTHOR("Peiheng Li, Chengrui Li, Yitong Bai");
196 MODULE_DESCRIPTION("Fruit Ninja DE1-SoC FPGA driver (ultrasonic)");
```

Listing 4: `software/top_module.c`: complete Linux platform/misc driver.

```c
/*
 * game.c -- Fruit-Ninja userspace game loop (ultrasonic + bombs)
 *
 * Level controls the number of simultaneous objects:
 *   Level 1 = 1 object, Level 2 = 2 objects, Level 3 = 3 objects.
 *
 * Game-state register (GAME_ST_WRITE) bits:
 *   bit0      = selected slot object is bomb
 *   bits 3:1  = fruit sprite id (0=apple, 1=banana, 2=watermelon)
 *   bit4      = show cut/blown sprite
 *   bits12:5  = 8-bit rotation angle (0..255 = one full turn)
 *   bits17:16 = object slot (0..2), consumed by the driver
 *   bits23:18 = combo display count, 0 hides combo text
 *   bits27:24 = combo x offset index, maps to roughly -32..+28 px
 *   bits31:28 = combo y offset index, maps to roughly -32..+28 px
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <time.h>
#include "top_module.h"

/* ---- Tuning ---- */
#define Q              4
#define VX_MAX_Q       40
#define LINE_Y         320
#define RADIUS_MIN     22
#define RADIUS_MAX     34
#define FRAME_US       33000
#define OBJECT_LIFETIME_FRAMES 60
#define CROSS_BASE_FRAME 24
#define CROSS_STAGGER_FRAMES 14
#define CROSS_JITTER_FRAMES 3
#define EXIT_Y         (SCREEN_H + 64)

#define NEAR_CM        20u
#define MAX_LIVES      3
#define MAX_OBJECTS    3
#define LEVEL2_SCORE   10
#define LEVEL3_SCORE   20
#define BOMB_PERCENT   30
#define BLOW_DISPLAY_US 1000000
#define BLOW_DISPLAY_FRAMES (BLOW_DISPLAY_US / FRAME_US)
#define COMBO_HEAL_STEP 10
#define NUM_FRUIT_SPRITES 3

#define ANGLE_FAST_INC_Q8 4369u
#define ANGLE_MED_INC_Q8  2185u
#define ANGLE_SLOW_INC_Q8 1456u

#define DEV_PATH       "/dev/top_module"

typedef struct {
    int x;
    int y_q;
    int vx_q;
    int vy_q;
    int start_y_q;
    int age_frames;
```

```c
    int radius;
    int visible;
    int is_bomb;
    int fruit_id;
    int blown;
    int blow_frames_left;
    int combo_display;
    int combo_dx_idx;
    int combo_dy_idx;
    int already_cut;
    unsigned int angle_q8;
    unsigned int angle_inc_q8;
} object_t;

static int fd;
static int score;
static int lives;
static int level;
static int combo_count;
static int active_objects;
static object_t objects[MAX_OBJECTS];
static volatile sig_atomic_t g_stop = 0;

static int prev_near;
static unsigned int last_sample_id;

static void on_sigint(int s) { (void)s; g_stop = 1; }

static int rand_range(int lo, int hi)
{
    return lo + rand() % (hi - lo + 1);
}

static int level_for_score(int s)
{
    if (s >= LEVEL3_SCORE) return 3;
    if (s >= LEVEL2_SCORE) return 2;
    return 1;
}

static void refresh_level(void)
{
    int new_level = level_for_score(score);
    if (new_level != level) {
        level = new_level;
        printf("  LEVEL UP!  level=%d\n", level);
        fflush(stdout);
    }
}

static void write_fruit_hw(int slot)
{
    object_t *obj = &objects[slot];
    int y_pixel = obj->y_q / Q;
    int x_pixel = obj->x;
    fruit_params_t fp;

    if (y_pixel < 0) y_pixel = 0;
    if (y_pixel > 1023) y_pixel = 1023;
    if (x_pixel < 0) x_pixel = 0;
    if (x_pixel > 1023) x_pixel = 1023;

    fp.slot = (unsigned int)slot;
    fp.x = (unsigned int)x_pixel;
    fp.y = (unsigned int)y_pixel;
    fp.radius = (unsigned int)obj->radius;
    fp.visible = (unsigned int)(obj->visible ? 1 : 0);
    ioctl(fd, FRUIT_WRITE, &fp);
```

```
}

static void write_score_hw(void)
{
    unsigned int s = (unsigned int)score;
    ioctl(fd, SCORE_WRITE, &s);
}

static void write_lives_hw(void)
{
    unsigned int l = (lives < 0) ? 0u : (unsigned int)lives;
    ioctl(fd, LIVES_WRITE, &l);
}

static void write_state_hw(int slot)
{
    object_t *obj = &objects[slot];
    unsigned int combo = (obj->combo_display > 63) ? 63u : (unsigned int)obj->combo_display;
    unsigned int s = (obj->is_bomb ? 1u : 0u)
                   | (((unsigned int)obj->fruit_id & 7u) << 1)
                   | (obj->blown ? (1u << 4) : 0u)
                   | (((obj->angle_q8 >> 8) & 0xFFu) << 5)
                   | (((unsigned int)slot & 3u) << 16)
                   | ((combo & 0x3Fu) << 18)
                   | (((unsigned int)obj->combo_dx_idx & 0xFu) << 24)
                   | (((unsigned int)obj->combo_dy_idx & 0xFu) << 28);
    ioctl(fd, GAME_ST_WRITE, &s);
}

static void spawn_object(int slot)
{
    object_t *obj = &objects[slot];
    int zone = rand() % 3;
    int target_cross_frame;

    memset(obj, 0, sizeof(*obj));
    obj->radius = rand_range(RADIUS_MIN, RADIUS_MAX);
    obj->visible = 1;
    obj->is_bomb = (rand() % 100) < BOMB_PERCENT;
    obj->fruit_id = rand() % NUM_FRUIT_SPRITES;
    obj->angle_q8 = (unsigned int)(rand() & 0xFFFF);

    switch (rand() % 3) {
    case 0:  obj->angle_inc_q8 = ANGLE_FAST_INC_Q8; break;
    case 1:  obj->angle_inc_q8 = ANGLE_MED_INC_Q8;  break;
    default: obj->angle_inc_q8 = ANGLE_SLOW_INC_Q8; break;
    }

    switch (zone) {
    case 0:
        obj->x = rand_range(SCREEN_W / 4, 3 * SCREEN_W / 4);
        obj->y_q = rand_range(10, 40) * Q;
        obj->vx_q = rand_range(-VX_MAX_Q / 2, VX_MAX_Q / 2);
        break;
    case 1:
        obj->x = rand_range(30, SCREEN_W / 4);
        obj->y_q = rand_range(10, LINE_Y / 2) * Q;
        obj->vx_q = rand_range(VX_MAX_Q / 2, VX_MAX_Q);
        break;
    default:
        obj->x = rand_range(3 * SCREEN_W / 4, SCREEN_W - 30);
        obj->y_q = rand_range(10, LINE_Y / 2) * Q;
        obj->vx_q = rand_range(-VX_MAX_Q, -VX_MAX_Q / 2);
        break;
    }

    obj->start_y_q = obj->y_q;
    target_cross_frame = CROSS_BASE_FRAME
```

```
                        + slot * CROSS_STAGGER_FRAMES
                        + rand_range(-CROSS_JITTER_FRAMES, CROSS_JITTER_FRAMES);
    if (target_cross_frame < 8)
        target_cross_frame = 8;
    if (target_cross_frame > OBJECT_LIFETIME_FRAMES - 6)
        target_cross_frame = OBJECT_LIFETIME_FRAMES - 6;
    obj->vy_q = ((LINE_Y * Q) - obj->start_y_q) / target_cross_frame;
    if (obj->vy_q < 1)
        obj->vy_q = 1;

    printf("  spawn slot=%d %s zone=%d sprite=%d @ x=%d y=%d vx=%+d vy=%+d r=%d cross=%d rot_inc=%u\n",
           slot, obj->is_bomb ? "BOMB" : "fruit", zone, obj->fruit_id,
           obj->x, obj->y_q / Q, obj->vx_q, obj->vy_q, obj->radius,
           target_cross_frame, obj->angle_inc_q8);
    fflush(stdout);
    write_state_hw(slot);
    write_fruit_hw(slot);
}

static void hide_object(int slot)
{
    objects[slot].visible = 0;
    objects[slot].blown = 0;
    objects[slot].blow_frames_left = 0;
    objects[slot].combo_display = 0;
    write_state_hw(slot);
    write_fruit_hw(slot);
}

static void spawn_wave(void)
{
    int i;

    refresh_level();
    active_objects = level;
    printf("  --- Level %d wave: %d object(s) ---\n", level, active_objects);
    for (i = 0; i < MAX_OBJECTS; i++) {
        if (i < active_objects) {
            spawn_object(i);
        } else {
            memset(&objects[i], 0, sizeof(objects[i]));
            hide_object(i);
        }
    }
}

static int wave_done(void)
{
    int i;

    for (i = 0; i < active_objects; i++) {
        if (objects[i].visible)
            return 0;
    }
    return 1;
}

static void update_object(int slot)
{
    object_t *obj = &objects[slot];
    int y_pixel;

    if (!obj->visible) return;

    if (obj->blown) {
        if (obj->blow_frames_left > 0)
            obj->blow_frames_left--;
        if (obj->blow_frames_left <= 0)
```

```
            hide_object(slot);
        return;
    }

    obj->age_frames++;
    obj->angle_q8 += obj->angle_inc_q8;
    obj->y_q = obj->start_y_q + obj->vy_q * obj->age_frames;
    obj->x += obj->vx_q / Q;
    y_pixel = obj->y_q / Q;

    if (y_pixel > EXIT_Y) {
        if (obj->visible && !obj->already_cut) {
            if (!obj->is_bomb) {
                lives--;
                combo_count = 0;
                write_lives_hw();
                printf("  MISS slot=%d fruit!  lives=%d combo reset\n", slot, lives);
                fflush(stdout);
            } else {
                printf("  slot=%d bomb dodged\n", slot);
            }
        }
        hide_object(slot);
        return;
    }

    if (obj->x < obj->radius) {
        obj->x = obj->radius;
        obj->vx_q = -obj->vx_q;
    }
    if (obj->x > SCREEN_W - obj->radius) {
        obj->x = SCREEN_W - obj->radius;
        obj->vx_q = -obj->vx_q;
    }
}

static int check_swipe(void)
{
    motion_status_t m;
    unsigned int cm;
    int is_new;
    int curr_near;
    int edge;

    if (ioctl(fd, MOTION_READ, &m) < 0) return 0;

    is_new = m.motion_detected || (m.frame_counter != last_sample_id);
    if (!is_new) return 0;
    last_sample_id = m.frame_counter;

    cm = m.changed_pixel_count / ULTRA_TICKS_PER_CM;
    curr_near = (cm > 0 && cm < NEAR_CM);
    edge = (!prev_near && curr_near);
    prev_near = curr_near;
    return edge;
}

static int object_is_cuttable(int slot)
{
    object_t *obj = &objects[slot];
    int y_pixel = obj->y_q / Q;
    int dy = y_pixel - LINE_Y;

    if (dy < 0) dy = -dy;
    return obj->visible && !obj->already_cut && dy <= (obj->radius + 24); // 24 is a fudge factor to
    make it easier to cut objects that are slightly above/below the line
}

```

```c
static void write_all_objects_hw(void)
{
    int i;

    for (i = 0; i < MAX_OBJECTS; i++) {
        write_state_hw(i);
        write_fruit_hw(i);
    }
}

int main(void)
{
    signal(SIGINT, on_sigint);
    srand((unsigned)time(NULL));

    fd = open(DEV_PATH, O_RDWR);
    if (fd < 0) {
        perror("open " DEV_PATH);
        return 1;
    }

    score = 0;
    lives = MAX_LIVES;
    level = 1;
    combo_count = 0;
    active_objects = 1;

    printf("============================================\n");
    printf("   FRUIT NINJA (ultrasonic + bombs)\n");
    printf("============================================\n");
    printf("Swipe within %u cm to cut the flying object.\n", NEAR_CM);
    printf("Fruit = +1 score and combo. Bomb or missed fruit resets combo.\n");
    printf("Level 1: 1 object, Level 2 at %d points: 2 objects, Level 3 at %d points: 3 objects.\n\n",
           LEVEL2_SCORE, LEVEL3_SCORE);

    write_score_hw();
    write_lives_hw();
    spawn_wave();

    while (!g_stop && lives > 0) {
        int i;

        for (i = 0; i < active_objects; i++)
            update_object(i);

        if (check_swipe()) {
            int cut_count = 0;
            int bomb_cut = 0;
            int cut_slots[MAX_OBJECTS];

            for (i = 0; i < active_objects; i++) {
                object_t *obj = &objects[i];

                if (!object_is_cuttable(i))
                    continue;

                obj->already_cut = 1;
                obj->blown = 1;
                obj->blow_frames_left = BLOW_DISPLAY_FRAMES;
                cut_slots[cut_count++] = i;

                if (obj->is_bomb) {
                    bomb_cut = 1;
                    lives--;
                    write_lives_hw();
                    printf("  BOMB CUT slot=%d!  lives=%d\n", i, lives);
                }
            }
```

```
404
405             if (bomb_cut) {
406                 combo_count = 0;
407                 for (i = 0; i < cut_count; i++)
408                     objects[cut_slots[i]].combo_display = 0;
409                 printf("  combo reset\n");
410             }
411
412             if (!bomb_cut) {
413                 for (i = 0; i < cut_count; i++) {
414                     object_t *obj = &objects[cut_slots[i]];
415
416                     if (obj->is_bomb)
417                         continue;
418
419                     score++;
420                     combo_count++;
421                     if (combo_count >= 2) {
422                         obj->combo_display = combo_count;
423                         obj->combo_dx_idx = rand_range(0, 15);
424                         obj->combo_dy_idx = rand_range(0, 15);
425                     }
426                     if ((combo_count % COMBO_HEAL_STEP) == 0 && lives < MAX_LIVES) {
427                         lives++;
428                         write_lives_hw();
429                         printf("  COMBO HEAL!  lives=%d\n", lives);
430                     }
431                     printf("  CUT fruit slot=%d!  score=%d combo=%d\n",
432                            cut_slots[i], score, combo_count);
433                     write_score_hw();
434                     refresh_level();
435                 }
436             } else {
437                 for (i = 0; i < cut_count; i++) {
438                     object_t *obj = &objects[cut_slots[i]];
439
440                     if (!obj->is_bomb) {
441                         score++;
442                         printf("  CUT fruit slot=%d!  score=%d combo=0\n",
443                                cut_slots[i], score);
444                         write_score_hw();
445                         refresh_level();
446                     }
447                 }
448             }
449
450             if (cut_count > 0)
451                 fflush(stdout);
452         }
453
454         if (lives > 0 && wave_done()) {
455             spawn_wave();
456             continue;
457         }
458
459         write_all_objects_hw();
460         usleep(FRAME_US);
461     }
462
463     memset(objects, 0, sizeof(objects));
464     lives = 0;
465     write_lives_hw();
466     write_all_objects_hw();
467
468     printf("\n==========================================\n");
469     printf("  GAME OVER.   Final score: %d\n", score);
470     printf("==========================================\n");
471
```

```
    close(fd);
    return 0;
}
```

Listing 5: `software/game.c`: complete user-space game implementation.

A.2 Complete RTL Directory

```
// top_module.sv
// Simplified Fruit-Ninja on DE1-SoC
//   - HC-SR04 ultrasonic sensor -> swipe detection
//   - VGA 640x480               -> game graphics (fruit + dashed line)
//   - Avalon-MM 32-bit slave    -> HPS register interface

module top_module (
    // ======= System =======
    input  logic        clk,       // 50 MHz
    input  logic        reset,

    // ======= Avalon-MM Slave =======
    input  logic        chipselect,
    input  logic [3:0]  address,   // word address
    input  logic        read,
    output logic [31:0] readdata,
    input  logic        write,
    input  logic [31:0] writedata,

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

    // ======= Debug =======
    output logic [9:0]  LED,

    // ======= Ultrasonic (HC-SR04) =======
    input  logic        echo_in,      // GPIO_0[0], echo from sensor (3.3V-level)
    output logic        trig_out      // GPIO_1[0], trigger to sensor
);

    // ============================================================
    //  Ultrasonic CSR sub-slave
    //  Top-level Avalon addresses 0..3 (bytes 0x00..0x0C) are forwarded
    //  to the ultra_sensor_csr Avalon slave; addresses 4..11 hit the
    //  game registers below.
    // ============================================================
    wire [31:0] ultra_readdata;
    wire        ultra_cs = chipselect & (address[3:2] == 2'b00);

    ultra_sensor_csr ultra_i (
        .clk            (clk),
        .reset_n        (~reset),
        .s0_address     (address[1:0]),
        .s0_read        (read),
        .s0_write       (write),
        .s0_chipselect  (ultra_cs),
        .s0_writedata   (writedata),
        .s0_readdata    (ultra_readdata),
        .s0_waitrequest (),
        .echo_in        (echo_in),
        .trig_out       (trig_out)
    );

    // Echo into clk domain for debug display
    reg echo_s1, echo_s2;
    always_ff @(posedge clk) begin
```

```
        echo_s1 <= echo_in;
        echo_s2 <= echo_s1;
    end

    // ============================================================
    //  VGA Timing Generator  (50 MHz -> 25 MHz pixel clock)
    // ============================================================
    logic [10:0] hcount;
    logic [ 9:0] vcount;
    logic        endOfField;

    vga_counter counter_0 (
        .clk50       (clk),
        .reset       (reset),
        .enable      (1'b1),
        .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),
        .endOfField  (endOfField)
    );

    // ============================================================
    //  Game Registers  (written by HPS via Avalon-MM)
    // ============================================================
    //  Word addr  | Name             | Dir
    //  0  (0x00)  | ultra_status     | R  bit0=new_sample, bit1=echo, bit2=trig
    //  1  (0x04)  | last_echo_cnt    | R
    //  2  (0x08)  | current_echo_cnt | R
    //  3  (0x0C)  | sample_id        | R
    //  4  (0x10)  | fruit_x          | W   selected object slot
    //  5  (0x14)  | fruit_y          | W   selected object slot
    //  6  (0x18)  | fruit_radius     | W   selected object slot
    //  7  (0x1C)  | fruit_ctrl       | W   selected object slot, bit0 = visible
    //  8  (0x20)  | score            | W
    //  9  (0x24)  | game_state       | W   selected object slot,
    //                                        bit0=bomb, bits[3:1]=fruit sprite id,
    //                                        bit4=blow, bits[12:5]=angle
    // 10  (0x28)  | lives            | W   0..3
    // 11  (0x2C)  | object_index     | W   selected object slot, 0..2

    localparam int NUM_OBJECTS = 3;

    reg [9:0]  fruit_x_reg      [0:NUM_OBJECTS-1];
    reg [9:0]  fruit_y_reg      [0:NUM_OBJECTS-1];
    reg [9:0]  fruit_radius_reg [0:NUM_OBJECTS-1];
    reg        fruit_visible_reg[0:NUM_OBJECTS-1];
    reg [31:0] score_reg        = 32'd0;
    reg [31:0] game_state_reg   [0:NUM_OBJECTS-1];
    reg [ 1:0] lives_reg        = 2'd3;
    reg [ 1:0] object_index_reg = 2'd0;

    wire [1:0] object_index_safe = (object_index_reg >= 2'd3) ? 2'd0 : object_index_reg;
    integer obj_i;
    integer si;

    // --- Avalon read ---
    always_ff @(posedge clk) begin
        if (chipselect && read) begin
            case (address)
                4'd0, 4'd1, 4'd2, 4'd3:
                          readdata <= ultra_readdata;
                4'd4:     readdata <= {22'd0, fruit_x_reg[object_index_safe]};
                4'd5:     readdata <= {22'd0, fruit_y_reg[object_index_safe]};
                4'd6:     readdata <= {22'd0, fruit_radius_reg[object_index_safe]};
```

```
                4'd7:    readdata <= {31'd0, fruit_visible_reg[object_index_safe]};
                4'd8:    readdata <= score_reg;
                4'd9:    readdata <= game_state_reg[object_index_safe];
                4'd10:   readdata <= {30'd0, lives_reg};
                4'd11:   readdata <= {30'd0, object_index_reg};
                default: readdata <= 32'd0;
            endcase
        end
    end

    // --- Avalon write --- (writes to 0..3 flow through ultra_cs to ultra_sensor_csr)
    always_ff @(posedge clk) begin
        if (reset) begin
            for (obj_i = 0; obj_i < NUM_OBJECTS; obj_i = obj_i + 1) begin
                fruit_x_reg[obj_i]       <= 10'd320;
                fruit_y_reg[obj_i]       <= 10'd240;
                fruit_radius_reg[obj_i]  <= 10'd30;
                fruit_visible_reg[obj_i] <= 1'b0;
                game_state_reg[obj_i]    <= 32'd0;
            end
            score_reg         <= 32'd0;
            lives_reg         <= 2'd3;
            object_index_reg  <= 2'd0;
        end else if (chipselect && write) begin
            case (address)
                4'd4: fruit_x_reg[object_index_safe]       <= writedata[9:0];
                4'd5: fruit_y_reg[object_index_safe]       <= writedata[9:0];
                4'd6: fruit_radius_reg[object_index_safe]  <= writedata[9:0];
                4'd7: fruit_visible_reg[object_index_safe] <= writedata[0];
                4'd8: score_reg         <= writedata;
                4'd9: game_state_reg[(writedata[17:16] >= 2'd3) ? object_index_safe : writedata[17:16]]
 <= writedata;
                4'd10: lives_reg        <= writedata[1:0];
                4'd11: object_index_reg <= (writedata[1:0] >= 2'd3) ? 2'd0 : writedata[1:0];
                default: ;
            endcase
        end
    end

    // ================================================================
    //  VGA Rendering  (64x64 RGBA4444 sprite ROM + registered output)
    // ================================================================
    wire [9:0] pixel_x = hcount[10:1];   // 0-639
    wire [9:0] pixel_y = vcount[9:0];    // 0-479

    // ---- Rotated 64x64 sprite addresses, centered at each object x/y ----
    wire signed [11:0] sprite_dx       [0:NUM_OBJECTS-1];
    wire signed [11:0] sprite_dy       [0:NUM_OBJECTS-1];
    wire               sprite_in_box   [0:NUM_OBJECTS-1];
    wire signed [11:0] cos_q10         [0:NUM_OBJECTS-1];
    wire signed [11:0] sin_q10         [0:NUM_OBJECTS-1];
    wire signed [23:0] cos_dx          [0:NUM_OBJECTS-1];
    wire signed [23:0] sin_dy          [0:NUM_OBJECTS-1];
    wire signed [23:0] cos_dy          [0:NUM_OBJECTS-1];
    wire signed [23:0] sin_dx          [0:NUM_OBJECTS-1];
    wire signed [24:0] rot_x_q10       [0:NUM_OBJECTS-1];
    wire signed [24:0] rot_y_q10       [0:NUM_OBJECTS-1];
    wire signed [24:0] src_x_q10       [0:NUM_OBJECTS-1];
    wire signed [24:0] src_y_q10       [0:NUM_OBJECTS-1];
    wire               sprite_src_valid[0:NUM_OBJECTS-1];
    wire [5:0]         sprite_src_x    [0:NUM_OBJECTS-1];
    wire [5:0]         sprite_src_y    [0:NUM_OBJECTS-1];
    wire [11:0]        sprite_addr     [0:NUM_OBJECTS-1];

    wire [15:0] apple_pixel            [0:NUM_OBJECTS-1];
    wire [15:0] apple_blow_pixel       [0:NUM_OBJECTS-1];
    wire [15:0] banana_pixel           [0:NUM_OBJECTS-1];
    wire [15:0] banana_blow_pixel      [0:NUM_OBJECTS-1];
```

```
    wire [15:0] watermelon_pixel       [0:NUM_OBJECTS-1];
    wire [15:0] watermelon_blow_pixel  [0:NUM_OBJECTS-1];
    wire [15:0] bomb_pixel             [0:NUM_OBJECTS-1];
    wire [15:0] boom_blow_pixel        [0:NUM_OBJECTS-1];

    genvar gi;
    generate
        for (gi = 0; gi < NUM_OBJECTS; gi = gi + 1) begin : sprite_slots
            assign sprite_dx[gi] =
                $signed({2'b00, pixel_x}) - $signed({2'b00, fruit_x_reg[gi]});
            assign sprite_dy[gi] =
                $signed({2'b00, pixel_y}) - $signed({2'b00, fruit_y_reg[gi]});
            assign sprite_in_box[gi] = fruit_visible_reg[gi]
                                    && (sprite_dx[gi] >= -12'sd46) && (sprite_dx[gi] <= 12'sd46)
                                    && (sprite_dy[gi] >= -12'sd46) && (sprite_dy[gi] <= 12'sd46);

            rot_lut rot_lut_i (
                .angle   (game_state_reg[gi][12:5]),
                .cos_q10 (cos_q10[gi]),
                .sin_q10 (sin_q10[gi])
            );

            assign cos_dx[gi] = cos_q10[gi] * sprite_dx[gi];
            assign sin_dy[gi] = sin_q10[gi] * sprite_dy[gi];
            assign cos_dy[gi] = cos_q10[gi] * sprite_dy[gi];
            assign sin_dx[gi] = sin_q10[gi] * sprite_dx[gi];
            assign rot_x_q10[gi] = {cos_dx[gi][23], cos_dx[gi]} + {sin_dy[gi][23], sin_dy[gi]};
            assign rot_y_q10[gi] = {cos_dy[gi][23], cos_dy[gi]} - {sin_dx[gi][23], sin_dx[gi]};
            assign src_x_q10[gi] = (25'sd32 <<< 10) + rot_x_q10[gi];
            assign src_y_q10[gi] = (25'sd32 <<< 10) + rot_y_q10[gi];
            assign sprite_src_valid[gi] = sprite_in_box[gi]
                                       && (src_x_q10[gi] >= 25'sd0) && (src_x_q10[gi] < 25'sd65536)
                                       && (src_y_q10[gi] >= 25'sd0) && (src_y_q10[gi] < 25'sd65536);
            assign sprite_src_x[gi] = src_x_q10[gi][15:10];
            assign sprite_src_y[gi] = src_y_q10[gi][15:10];
            assign sprite_addr[gi] = sprite_src_valid[gi] ? {sprite_src_y[gi], sprite_src_x[gi]} : 12'
    d0;

            sprite_rom #(.INIT_FILE("../pics/apple.hex")) apple_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (apple_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/apple_blow.hex")) apple_blow_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (apple_blow_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/banana.hex")) banana_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (banana_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/banana_blow.hex")) banana_blow_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (banana_blow_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/watermelon.hex")) watermelon_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (watermelon_pixel[gi])
            );

```

```
            sprite_rom #(.INIT_FILE("../pics/watermelon_blow.hex")) watermelon_blow_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (watermelon_blow_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/bomb.hex")) bomb_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (bomb_pixel[gi])
            );

            sprite_rom #(.INIT_FILE("../pics/boom_blow.hex")) boom_blow_rom (
                .clk   (clk),
                .addr  (sprite_addr[gi]),
                .pixel (boom_blow_pixel[gi])
            );
        end
    endgenerate

    // ---- Repeating 128x96 background made from twelve 32x32 RGB888 tiles ----
    wire [6:0] bg_x_mod_128 = pixel_x[6:0];
    wire [1:0] bg_tile_col = bg_x_mod_128[6:5]; // 0..3 for 4 tile columns
    wire [4:0] bg_local_x = bg_x_mod_128[4:0]; // 0..31 pixel x within tile

    wire [9:0] bg_y_minus_384 = pixel_y - 10'd384; // Avoid negative numbers for modulo: 0..383 map to themselves, 384..479 map to 0..95
    wire [9:0] bg_y_minus_288 = pixel_y - 10'd288;
    wire [9:0] bg_y_minus_192 = pixel_y - 10'd192;
    wire [9:0] bg_y_minus_96  = pixel_y - 10'd96;
    wire [6:0] bg_y_mod_96 =
        (pixel_y >= 10'd384) ? bg_y_minus_384[6:0] :
        (pixel_y >= 10'd288) ? bg_y_minus_288[6:0] :
        (pixel_y >= 10'd192) ? bg_y_minus_192[6:0] :
        (pixel_y >= 10'd96)  ? bg_y_minus_96[6:0]  :
                               pixel_y[6:0];
    wire [1:0] bg_tile_row = (bg_y_mod_96 >= 7'd64) ? 2'd2 :
                             (bg_y_mod_96 >= 7'd32) ? 2'd1 : 2'd0; // 0..2 for 3 tile rows
    wire [4:0] bg_local_y = bg_y_mod_96[4:0]; // 0..31 pixel y within tile
    wire [3:0] bg_tile_id = {2'b00, bg_tile_col}
                          + (bg_tile_row == 2'd1 ? 4'd4 :
                             bg_tile_row == 2'd2 ? 4'd8 : 4'd0); // 0..11 tile id for 12 unique tiles in the background
    wire [9:0] bg_tile_addr = {bg_local_y, bg_local_x}; // 32x32 tiles, so 5 bits each for x and y within tile

    wire [23:0] bg_tile_0000_pixel;
    wire [23:0] bg_tile_0001_pixel;
    wire [23:0] bg_tile_0002_pixel;
    wire [23:0] bg_tile_0003_pixel;
    wire [23:0] bg_tile_0004_pixel;
    wire [23:0] bg_tile_0005_pixel;
    wire [23:0] bg_tile_0006_pixel;
    wire [23:0] bg_tile_0007_pixel;
    wire [23:0] bg_tile_0008_pixel;
    wire [23:0] bg_tile_0009_pixel;
    wire [23:0] bg_tile_0010_pixel;
    wire [23:0] bg_tile_0011_pixel;

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0000.hex")) bg_tile_0000_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0000_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0001.hex")) bg_tile_0001_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
```

```
        .pixel (bg_tile_0001_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0002.hex")) bg_tile_0002_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0002_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0003.hex")) bg_tile_0003_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0003_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0004.hex")) bg_tile_0004_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0004_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0005.hex")) bg_tile_0005_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0005_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0006.hex")) bg_tile_0006_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0006_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0007.hex")) bg_tile_0007_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0007_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0008.hex")) bg_tile_0008_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0008_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0009.hex")) bg_tile_0009_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0009_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0010.hex")) bg_tile_0010_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0010_pixel)
    );

    tile_rom_rgb888 #(.INIT_FILE("../pics/tiles_32_rgb888_hex/0011.hex")) bg_tile_0011_rom (
        .clk   (clk),
        .addr  (bg_tile_addr),
        .pixel (bg_tile_0011_pixel)
    );

    // ---- Dashed line (horizontal, y = 320, thickness = 2 px each side) ----
    localparam [9:0] LINE_Y = 10'd320;
    wire near_line     = (pixel_y >= (LINE_Y - 10'd2)) && (pixel_y <= (LINE_Y + 10'd2));
    wire on_dashed_line = near_line && ~pixel_x[4];  // 16-on / 16-off

```

```
    // ---- Heart lives display (top-right, 3 hearts) ----
    localparam [9:0] H_Y  = 10'd6;
    localparam [9:0] H0_X = 10'd556;
    localparam [9:0] H1_X = 10'd584;
    localparam [9:0] H2_X = 10'd612;

    wire signed [10:0] heart_dy     = $signed({1'b0, pixel_y}) - 11'sd12;
    wire signed [21:0] heart_dysq_s = heart_dy * heart_dy;
    wire        [21:0] heart_dysq   = heart_dysq_s[21:0];
    wire [9:0] h_toff = (pixel_y >= H_Y + 10'd6) ? (pixel_y - (H_Y + 10'd6)) : 10'd0;

    wire signed [10:0] h0_dxl = $signed({1'b0, pixel_x}) - 11'sd562;
    wire signed [10:0] h0_dxr = $signed({1'b0, pixel_x}) - 11'sd573;
    wire signed [21:0] h0_dxl_sq_s = h0_dxl * h0_dxl;
    wire signed [21:0] h0_dxr_sq_s = h0_dxr * h0_dxr;
    wire h0_shape =
        (pixel_x >= H0_X) && (pixel_x <= H0_X + 10'd23) &&
        (pixel_y >= H_Y)  && (pixel_y <= H_Y  + 10'd17) &&
        ( ((pixel_y <= H_Y + 10'd6) && (h0_dxl_sq_s[21:0] + heart_dysq <= 22'd36))
        | ((pixel_y <= H_Y + 10'd6) && (h0_dxr_sq_s[21:0] + heart_dysq <= 22'd36))
        | (pixel_y >= H_Y + 10'd6 && pixel_y <= H_Y + 10'd17 &&
           pixel_x >= H0_X + h_toff &&
           pixel_x <= H0_X + 10'd23 - h_toff) );

    wire signed [10:0] h1_dxl = $signed({1'b0, pixel_x}) - 11'sd590;
    wire signed [10:0] h1_dxr = $signed({1'b0, pixel_x}) - 11'sd601;
    wire signed [21:0] h1_dxl_sq_s = h1_dxl * h1_dxl;
    wire signed [21:0] h1_dxr_sq_s = h1_dxr * h1_dxr;
    wire h1_shape =
        (pixel_x >= H1_X) && (pixel_x <= H1_X + 10'd23) &&
        (pixel_y >= H_Y)  && (pixel_y <= H_Y  + 10'd17) &&
        ( ((pixel_y <= H_Y + 10'd6) && (h1_dxl_sq_s[21:0] + heart_dysq <= 22'd36))
        | ((pixel_y <= H_Y + 10'd6) && (h1_dxr_sq_s[21:0] + heart_dysq <= 22'd36))
        | (pixel_y >= H_Y + 10'd6 && pixel_y <= H_Y + 10'd17 &&
           pixel_x >= H1_X + h_toff &&
           pixel_x <= H1_X + 10'd23 - h_toff) );

    wire signed [10:0] h2_dxl = $signed({1'b0, pixel_x}) - 11'sd618;
    wire signed [10:0] h2_dxr = $signed({1'b0, pixel_x}) - 11'sd629;
    wire signed [21:0] h2_dxl_sq_s = h2_dxl * h2_dxl;
    wire signed [21:0] h2_dxr_sq_s = h2_dxr * h2_dxr;
    wire h2_shape =
        (pixel_x >= H2_X) && (pixel_x <= H2_X + 10'd23) &&
        (pixel_y >= H_Y)  && (pixel_y <= H_Y  + 10'd17) &&
        ( ((pixel_y <= H_Y + 10'd6) && (h2_dxl_sq_s[21:0] + heart_dysq <= 22'd36))
        | ((pixel_y <= H_Y + 10'd6) && (h2_dxr_sq_s[21:0] + heart_dysq <= 22'd36))
        | (pixel_y >= H_Y + 10'd6 && pixel_y <= H_Y + 10'd17 &&
           pixel_x >= H2_X + h_toff &&
           pixel_x <= H2_X + 10'd23 - h_toff) );

    wire in_heart = h0_shape | h1_shape | h2_shape;
    wire heart_is_active = (h0_shape & (lives_reg >= 2'd1))
                         | (h1_shape & (lives_reg >= 2'd2))
                         | (h2_shape & (lives_reg == 2'd3));

    function automatic logic [4:0] digit_row_bits(
        input logic [3:0] digit,
        input logic [2:0] row
    );
        begin
            case (digit)
                4'd0: case (row)
                    3'd0: digit_row_bits = 5'b11111;
                    3'd1: digit_row_bits = 5'b10001;
                    3'd2: digit_row_bits = 5'b10011;
                    3'd3: digit_row_bits = 5'b10101;
                    3'd4: digit_row_bits = 5'b11001;
                    3'd5: digit_row_bits = 5'b10001;
```

```
                    default: digit_row_bits = 5'b11111;
                endcase
                4'd1: case (row)
                    3'd0: digit_row_bits = 5'b00100;
                    3'd1: digit_row_bits = 5'b01100;
                    3'd2: digit_row_bits = 5'b00100;
                    3'd3: digit_row_bits = 5'b00100;
                    3'd4: digit_row_bits = 5'b00100;
                    3'd5: digit_row_bits = 5'b00100;
                    default: digit_row_bits = 5'b01110;
                endcase
                4'd2: case (row)
                    3'd0: digit_row_bits = 5'b11110;
                    3'd1: digit_row_bits = 5'b00001;
                    3'd2: digit_row_bits = 5'b00001;
                    3'd3: digit_row_bits = 5'b11110;
                    3'd4: digit_row_bits = 5'b10000;
                    3'd5: digit_row_bits = 5'b10000;
                    default: digit_row_bits = 5'b11111;
                endcase
                4'd3: case (row)
                    3'd0: digit_row_bits = 5'b11110;
                    3'd1: digit_row_bits = 5'b00001;
                    3'd2: digit_row_bits = 5'b00001;
                    3'd3: digit_row_bits = 5'b01110;
                    3'd4: digit_row_bits = 5'b00001;
                    3'd5: digit_row_bits = 5'b00001;
                    default: digit_row_bits = 5'b11110;
                endcase
                4'd4: case (row)
                    3'd0: digit_row_bits = 5'b10010;
                    3'd1: digit_row_bits = 5'b10010;
                    3'd2: digit_row_bits = 5'b10010;
                    3'd3: digit_row_bits = 5'b11111;
                    3'd4: digit_row_bits = 5'b00010;
                    3'd5: digit_row_bits = 5'b00010;
                    default: digit_row_bits = 5'b00010;
                endcase
                4'd5: case (row)
                    3'd0: digit_row_bits = 5'b11111;
                    3'd1: digit_row_bits = 5'b10000;
                    3'd2: digit_row_bits = 5'b10000;
                    3'd3: digit_row_bits = 5'b11110;
                    3'd4: digit_row_bits = 5'b00001;
                    3'd5: digit_row_bits = 5'b00001;
                    default: digit_row_bits = 5'b11110;
                endcase
                4'd6: case (row)
                    3'd0: digit_row_bits = 5'b01111;
                    3'd1: digit_row_bits = 5'b10000;
                    3'd2: digit_row_bits = 5'b10000;
                    3'd3: digit_row_bits = 5'b11110;
                    3'd4: digit_row_bits = 5'b10001;
                    3'd5: digit_row_bits = 5'b10001;
                    default: digit_row_bits = 5'b01110;
                endcase
                4'd7: case (row)
                    3'd0: digit_row_bits = 5'b11111;
                    3'd1: digit_row_bits = 5'b00001;
                    3'd2: digit_row_bits = 5'b00010;
                    3'd3: digit_row_bits = 5'b00100;
                    3'd4: digit_row_bits = 5'b01000;
                    3'd5: digit_row_bits = 5'b01000;
                    default: digit_row_bits = 5'b01000;
                endcase
                4'd8: case (row)
                    3'd0: digit_row_bits = 5'b01110;
                    3'd1: digit_row_bits = 5'b10001;
```

```
                    3'd2: digit_row_bits = 5'b10001;
                    3'd3: digit_row_bits = 5'b01110;
                    3'd4: digit_row_bits = 5'b10001;
                    3'd5: digit_row_bits = 5'b10001;
                    default: digit_row_bits = 5'b01110;
                endcase
                4'd9: case (row)
                    3'd0: digit_row_bits = 5'b01110;
                    3'd1: digit_row_bits = 5'b10001;
                    3'd2: digit_row_bits = 5'b10001;
                    3'd3: digit_row_bits = 5'b01111;
                    3'd4: digit_row_bits = 5'b00001;
                    3'd5: digit_row_bits = 5'b00001;
                    default: digit_row_bits = 5'b11110;
                endcase
                default: digit_row_bits = 5'b00000;
            endcase
        end
    endfunction

    function automatic logic [4:0] level_text_row_bits(
        input logic [2:0] char_idx,
        input logic [3:0] level_digit,
        input logic [2:0] row
    );
        begin
            case (char_idx)
                3'd0: case (row) // L
                    3'd0, 3'd1, 3'd2, 3'd3, 3'd4, 3'd5: level_text_row_bits = 5'b10000;
                    default: level_text_row_bits = 5'b11111;
                endcase
                3'd1: case (row) // E
                    3'd0: level_text_row_bits = 5'b11111;
                    3'd1: level_text_row_bits = 5'b10000;
                    3'd2: level_text_row_bits = 5'b10000;
                    3'd3: level_text_row_bits = 5'b11110;
                    3'd4: level_text_row_bits = 5'b10000;
                    3'd5: level_text_row_bits = 5'b10000;
                    default: level_text_row_bits = 5'b11111;
                endcase
                3'd2: case (row) // V
                    3'd0: level_text_row_bits = 5'b10001;
                    3'd1: level_text_row_bits = 5'b10001;
                    3'd2: level_text_row_bits = 5'b10001;
                    3'd3: level_text_row_bits = 5'b10001;
                    3'd4: level_text_row_bits = 5'b10001;
                    3'd5: level_text_row_bits = 5'b01010;
                    default: level_text_row_bits = 5'b00100;
                endcase
                3'd3: case (row) // E
                    3'd0: level_text_row_bits = 5'b11111;
                    3'd1: level_text_row_bits = 5'b10000;
                    3'd2: level_text_row_bits = 5'b10000;
                    3'd3: level_text_row_bits = 5'b11110;
                    3'd4: level_text_row_bits = 5'b10000;
                    3'd5: level_text_row_bits = 5'b10000;
                    default: level_text_row_bits = 5'b11111;
                endcase
                3'd4: case (row) // L
                    3'd0, 3'd1, 3'd2, 3'd3, 3'd4, 3'd5: level_text_row_bits = 5'b10000;
                    default: level_text_row_bits = 5'b11111;
                endcase
                3'd5: level_text_row_bits = 5'b00000; // space
                default: level_text_row_bits = digit_row_bits(level_digit, row);
            endcase
        end
    endfunction

```

```
    // ---- Numeric score display (top-left, 5x7 digits scaled 3x) ----
    localparam [9:0] SCORE_X = 10'd10;
    localparam [9:0] SCORE_Y = 10'd10;
    wire [9:0] score_value = (score_reg > 32'd999) ? 10'd999 : score_reg[9:0];
    wire [3:0] score_hundreds = score_value / 10'd100;
    wire [3:0] score_tens     = (score_value % 10'd100) / 10'd10;
    wire [3:0] score_ones     = score_value % 10'd10;

    wire in_score_area = (pixel_x >= SCORE_X) && (pixel_x < SCORE_X + 10'd51)
                      && (pixel_y >= SCORE_Y) && (pixel_y < SCORE_Y + 10'd21);
    wire [9:0] score_rel_x = pixel_x - SCORE_X;
    wire [9:0] score_rel_y = pixel_y - SCORE_Y;
    wire score_in_hundreds = in_score_area && (score_rel_x < 10'd15);
    wire score_in_tens     = in_score_area && (score_rel_x >= 10'd18) && (score_rel_x < 10'd33);
    wire score_in_ones     = in_score_area && (score_rel_x >= 10'd36);
    wire score_in_digit_box = score_in_hundreds | score_in_tens | score_in_ones;
    wire [9:0] score_digit_x = score_in_hundreds ? score_rel_x :
                               score_in_tens     ? (score_rel_x - 10'd18) :
                               score_in_ones     ? (score_rel_x - 10'd36) :
                                                   10'd0;
    wire [3:0] score_digit = score_in_hundreds ? score_hundreds :
                             score_in_tens     ? score_tens :
                                                 score_ones;
    wire score_digit_visible = score_in_digit_box
                             && (score_in_ones
                              || (score_in_tens && (score_hundreds != 4'd0 || score_tens != 4'd0))
                              || (score_in_hundreds && (score_hundreds != 4'd0)));
    wire [2:0] score_font_col = score_digit_x / 10'd3;
    wire [2:0] score_font_row = score_rel_y / 10'd3;
    wire [4:0] score_font_bits = digit_row_bits(score_digit, score_font_row);
    wire in_score_digit = score_digit_visible && score_font_bits[4 - score_font_col];

    // ---- Level display (top-center, same 5x7 font scaled 3x) ----
    localparam [9:0] LEVEL_X = 10'd259;
    localparam [9:0] LEVEL_Y = 10'd10;
    wire [3:0] level_value = (score_reg >= 32'd20) ? 4'd3 :
                             (score_reg >= 32'd10) ? 4'd2 : 4'd1;
    wire in_level_area = (pixel_x >= LEVEL_X) && (pixel_x < LEVEL_X + 10'd123)
                      && (pixel_y >= LEVEL_Y) && (pixel_y < LEVEL_Y + 10'd21);
    wire [9:0] level_rel_x = pixel_x - LEVEL_X;
    wire [9:0] level_rel_y = pixel_y - LEVEL_Y;
    wire [2:0] level_char_idx = level_rel_x / 10'd18;
    wire [9:0] level_char_x = level_rel_x - ({7'd0, level_char_idx} * 10'd18);
    wire level_in_glyph = in_level_area && (level_char_idx < 3'd7) && (level_char_x < 10'd15);
    wire [2:0] level_font_col = level_char_x / 10'd3;
    wire [2:0] level_font_row = level_rel_y / 10'd3;
    wire [4:0] level_font_bits = level_text_row_bits(level_char_idx, level_value, level_font_row);
    wire in_level_text = level_in_glyph && level_font_bits[4 - level_font_col];

    function automatic logic [4:0] combo_x_row_bits(input logic [2:0] row);
        begin
            case (row)
                3'd0: combo_x_row_bits = 5'b10001;
                3'd1: combo_x_row_bits = 5'b01010;
                3'd2: combo_x_row_bits = 5'b00100;
                3'd3: combo_x_row_bits = 5'b00100;
                3'd4: combo_x_row_bits = 5'b01010;
                3'd5: combo_x_row_bits = 5'b10001;
                default: combo_x_row_bits = 5'b10001;
            endcase
        end
    endfunction

    function automatic logic signed [10:0] combo_offset_px(input logic [3:0] idx);
        begin
            case (idx)
                4'd0:    combo_offset_px = -11'sd88;
                4'd1:    combo_offset_px = -11'sd80;
```

```
                4'd2:    combo_offset_px = -11'sd72;
                4'd3:    combo_offset_px = -11'sd64;
                4'd4:    combo_offset_px = -11'sd56;
                4'd5:    combo_offset_px = -11'sd48;
                4'd6:    combo_offset_px =  11'sd48;
                4'd7:    combo_offset_px =  11'sd56;
                4'd8:    combo_offset_px =  11'sd64;
                4'd9:    combo_offset_px =  11'sd72;
                4'd10:   combo_offset_px =  11'sd80;
                4'd11:   combo_offset_px =  11'sd88;
                4'd12:   combo_offset_px = -11'sd76;
                4'd13:   combo_offset_px = -11'sd60;
                4'd14:   combo_offset_px =  11'sd60;
                default: combo_offset_px =  11'sd76;
            endcase
        end
    endfunction

    // ---- Combo display near each cut fruit (x2, x3, ...), same 5x7 font scaled 3x ----
    wire [NUM_OBJECTS-1:0] in_combo_text;

    generate
        for (gi = 0; gi < NUM_OBJECTS; gi = gi + 1) begin : combo_text_slots
            wire [5:0] combo_value = (game_state_reg[gi][23:18] > 6'd63) ? 6'd63 : game_state_reg[gi
    ][23:18];
            wire signed [10:0] combo_dx_px = combo_offset_px(game_state_reg[gi][27:24]);
            wire signed [10:0] combo_dy_px = combo_offset_px(game_state_reg[gi][31:28]);
            wire signed [11:0] combo_x0_s = $signed({2'b00, fruit_x_reg[gi]}) + combo_dx_px;
            wire signed [11:0] combo_y0_s = $signed({2'b00, fruit_y_reg[gi]}) + combo_dy_px;
            wire [9:0] combo_x0 = (combo_x0_s < 12'sd0) ? 10'd0 :
                                  (combo_x0_s > 12'sd590) ? 10'd590 : combo_x0_s[9:0];
            wire [9:0] combo_y0 = (combo_y0_s < 12'sd0) ? 10'd0 :
                                  (combo_y0_s > 12'sd456) ? 10'd456 : combo_y0_s[9:0];
            wire combo_two_digits = combo_value >= 6'd10;
            wire [9:0] combo_w = combo_two_digits ? 10'd51 : 10'd33;
            wire combo_area = (combo_value >= 6'd2)
                           && (pixel_x >= combo_x0) && (pixel_x < combo_x0 + combo_w)
                           && (pixel_y >= combo_y0) && (pixel_y < combo_y0 + 10'd21);
            wire [9:0] combo_rel_x = pixel_x - combo_x0;
            wire [9:0] combo_rel_y = pixel_y - combo_y0;
            wire [1:0] combo_char_idx = combo_rel_x / 10'd18;
            wire [9:0] combo_char_x = combo_rel_x - ({8'd0, combo_char_idx} * 10'd18);
            wire combo_in_glyph = combo_area && (combo_char_x < 10'd15)
                                && ((!combo_two_digits && combo_char_idx < 2'd2)
                                 || ( combo_two_digits && combo_char_idx < 2'd3));
            wire [2:0] combo_font_col = combo_char_x / 10'd3;
            wire [2:0] combo_font_row = combo_rel_y / 10'd3;
            wire [3:0] combo_tens = combo_value / 6'd10;
            wire [3:0] combo_ones = combo_value % 6'd10;
            wire [3:0] combo_digit = combo_two_digits
                                   ? (combo_char_idx == 2'd1 ? combo_tens : combo_ones)
                                   : combo_ones;
            wire [4:0] combo_bits = (combo_char_idx == 2'd0)
                                   ? combo_x_row_bits(combo_font_row)
                                   : digit_row_bits(combo_digit, combo_font_row);
            assign in_combo_text[gi] = combo_in_glyph && combo_bits[4 - combo_font_col];
        end
    endgenerate

    // Delay non-ROM classification by one cycle to match synchronous M10K output.
    logic        blank_d;
    logic        in_heart_d;
    logic        heart_is_active_d;
    logic        in_score_digit_d;
    logic        in_level_text_d;
    logic        in_combo_text_d;
    logic        on_dashed_line_d;
    logic        sprite_in_box_d   [0:NUM_OBJECTS-1];
```

```
    logic        sprite_src_valid_d[0:NUM_OBJECTS-1];
    logic [4:0]  game_state_d      [0:NUM_OBJECTS-1];
    logic [3:0]  bg_tile_id_d;

    always_ff @(posedge clk) begin
        if (reset) begin
            blank_d          <= 1'b0;
            in_heart_d       <= 1'b0;
            heart_is_active_d <= 1'b0;
            in_score_digit_d <= 1'b0;
            in_level_text_d  <= 1'b0;
            in_combo_text_d  <= 1'b0;
            on_dashed_line_d <= 1'b0;
            bg_tile_id_d     <= 4'd0;
            for (obj_i = 0; obj_i < NUM_OBJECTS; obj_i = obj_i + 1) begin
                sprite_in_box_d[obj_i]    <= 1'b0;
                sprite_src_valid_d[obj_i] <= 1'b0;
                game_state_d[obj_i]       <= 5'd0;
            end
        end else begin
            blank_d          <= VGA_BLANK_N;
            in_heart_d       <= in_heart;
            heart_is_active_d <= heart_is_active;
            in_score_digit_d <= in_score_digit;
            in_level_text_d  <= in_level_text;
            in_combo_text_d  <= |in_combo_text;
            on_dashed_line_d <= on_dashed_line;
            bg_tile_id_d     <= bg_tile_id;
            for (obj_i = 0; obj_i < NUM_OBJECTS; obj_i = obj_i + 1) begin
                sprite_in_box_d[obj_i]    <= sprite_in_box[obj_i];
                sprite_src_valid_d[obj_i] <= sprite_src_valid[obj_i];
                game_state_d[obj_i]       <= game_state_reg[obj_i][4:0];
            end
        end
    end

    logic [15:0] sprite_pixel      [0:NUM_OBJECTS-1];
    logic [23:0] bg_pixel;
    // Determine which sprite pixel to show for each object slot, based on the delayed game state.
    always_comb begin
        for (si = 0; si < NUM_OBJECTS; si = si + 1) begin
            if (game_state_d[si][0]) begin
                sprite_pixel[si] = game_state_d[si][4] ? boom_blow_pixel[si] : bomb_pixel[si];
            end else begin
                case (game_state_d[si][3:1])
                    3'd1:    sprite_pixel[si] = game_state_d[si][4] ? banana_blow_pixel[si] :
    banana_pixel[si];
                    3'd2:    sprite_pixel[si] = game_state_d[si][4] ? watermelon_blow_pixel[si] :
    watermelon_pixel[si];
                    default: sprite_pixel[si] = game_state_d[si][4] ? apple_blow_pixel[si] :
    apple_pixel[si];
                endcase
            end
        end
    end

    wire [NUM_OBJECTS-1:0] sprite_opaque;
    assign sprite_opaque[0] = sprite_in_box_d[0] && sprite_src_valid_d[0] && (sprite_pixel[0][3:0] !=
    4'h0);
    assign sprite_opaque[1] = sprite_in_box_d[1] && sprite_src_valid_d[1] && (sprite_pixel[1][3:0] !=
    4'h0);
    assign sprite_opaque[2] = sprite_in_box_d[2] && sprite_src_valid_d[2] && (sprite_pixel[2][3:0] !=
    4'h0);

    logic [15:0] visible_sprite_pixel;

    always_comb begin
        if (sprite_opaque[2]) begin
```

```
            visible_sprite_pixel = sprite_pixel[2];
        end else if (sprite_opaque[1]) begin
            visible_sprite_pixel = sprite_pixel[1];
        end else begin
            visible_sprite_pixel = sprite_pixel[0];
        end
    end

    always_comb begin
        case (bg_tile_id_d)
            4'd0:    bg_pixel = bg_tile_0000_pixel;
            4'd1:    bg_pixel = bg_tile_0001_pixel;
            4'd2:    bg_pixel = bg_tile_0002_pixel;
            4'd3:    bg_pixel = bg_tile_0003_pixel;
            4'd4:    bg_pixel = bg_tile_0004_pixel;
            4'd5:    bg_pixel = bg_tile_0005_pixel;
            4'd6:    bg_pixel = bg_tile_0006_pixel;
            4'd7:    bg_pixel = bg_tile_0007_pixel;
            4'd8:    bg_pixel = bg_tile_0008_pixel;
            4'd9:    bg_pixel = bg_tile_0009_pixel;
            4'd10:   bg_pixel = bg_tile_0010_pixel;
            default: bg_pixel = bg_tile_0011_pixel;
        endcase
    end

    // ---- Pixel output mux ----
    // heart > combo > score/level > sprite > dashed line > background
    always_ff @(posedge clk) begin
        if (reset) begin
            VGA_R <= 8'd0;  VGA_G <= 8'd0;  VGA_B <= 8'd0;
        end else if (blank_d) begin
            if (in_heart_d) begin
                if (heart_is_active_d) begin
                    VGA_R <= 8'hFF;  VGA_G <= 8'h20;  VGA_B <= 8'h20;
                end else begin
                    VGA_R <= 8'h50;  VGA_G <= 8'h50;  VGA_B <= 8'h50;
                end
            end else if (in_combo_text_d) begin
                VGA_R <= 8'hFF;  VGA_G <= 8'hE0;  VGA_B <= 8'h40;
            end else if (in_score_digit_d || in_level_text_d) begin
                VGA_R <= 8'hFF;  VGA_G <= 8'hFF;  VGA_B <= 8'hFF;
            end else if (|sprite_opaque) begin
                // RGBA4444 stores the original RGBA8888 channel high nibbles.
                VGA_R <= {visible_sprite_pixel[15:12], visible_sprite_pixel[15:12]};
                VGA_G <= {visible_sprite_pixel[11:8],  visible_sprite_pixel[11:8]};
                VGA_B <= {visible_sprite_pixel[7:4],   visible_sprite_pixel[7:4]};
            end else if (on_dashed_line_d) begin
                // cut line -> white
                VGA_R <= 8'hFF;  VGA_G <= 8'hFF;  VGA_B <= 8'hFF;
            end else begin
                VGA_R <= bg_pixel[23:16];
                VGA_G <= bg_pixel[15:8];
                VGA_B <= bg_pixel[7:0];
            end
        end else begin
            VGA_R <= 8'd0;  VGA_G <= 8'd0;  VGA_B <= 8'd0;
        end
    end

    // ============================================================
    //  Debug LEDs
    // ============================================================
    assign LED[0]   = trig_out;          // trigger pulse (~20 Hz blink)
    assign LED[1]   = echo_s2;           // echo activity
    assign LED[9:2] = 8'b0;

endmodule
```

Listing 6: `rtl/top_module.sv`: complete custom peripheral top level, Avalon register file, VGA renderer, sprite/tile logic, HUD, and LEDs.

```
module ultra_sensor_csr (
    input  wire        clk,
    input  wire        reset_n,

    // Avalon-MM slave
    input  wire [1:0]  s0_address,
    input  wire        s0_read,
    input  wire        s0_write,
    input  wire        s0_chipselect,
    input  wire [31:0] s0_writedata,
    output reg  [31:0] s0_readdata,
    output wire        s0_waitrequest,

    // sensor pins
    input  wire        echo_in,
    output wire        trig_out
);

    assign s0_waitrequest = 1'b0;

    // ----------------------------
    // 1) trigger generation
    // 50MHz -> about every 100ms do one measurement
    // trig high about 100us
    // ----------------------------
    reg [22:0] period_cnt;
    wire period_cnt_full = (period_cnt == 23'd5000000);

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n)
            period_cnt <= 23'd0;
        else if (period_cnt_full)
            period_cnt <= 23'd0;
        else
            period_cnt <= period_cnt + 23'd1;
    end

    wire trig = (period_cnt > 23'd100) && (period_cnt < 23'd5100);
    assign trig_out = trig;

    // ----------------------------
    // 2) sync echo into FPGA clock domain
    // ----------------------------
    reg echo_ff1, echo_ff2, echo_prev;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            echo_ff1  <= 1'b0;
            echo_ff2  <= 1'b0;
            echo_prev <= 1'b0;
        end else begin
            echo_ff1  <= echo_in;
            echo_ff2  <= echo_ff1;
            echo_prev <= echo_ff2;
        end
    end

    wire echo_rise =  echo_ff2 & ~echo_prev;
    wire echo_fall = ~echo_ff2 &  echo_prev;

    // ----------------------------
    // 3) echo pulse width count
    // ----------------------------
```

```
    reg [31:0] echo_cnt;
    reg [31:0] last_echo_cnt;
    reg [31:0] sample_id;
    reg        new_sample;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            echo_cnt      <= 32'd0;
            last_echo_cnt <= 32'd0;
            sample_id     <= 32'd0;
            new_sample    <= 1'b0;
        end else begin
            // HPS write 1 to clear new_sample
            if (s0_chipselect && s0_write && (s0_address == 2'd0) && s0_writedata[0])
                new_sample <= 1'b0;

            if (trig) begin
                echo_cnt <= 32'd0;
            end else if (echo_ff2) begin
                echo_cnt <= echo_cnt + 32'd1;
            end

            if (echo_fall) begin
                last_echo_cnt <= echo_cnt;
                sample_id     <= sample_id + 32'd1;
                new_sample    <= 1'b1;
            end
        end
    end

    // ------------------------------
    // 4) register map
    // 0x0 status: bit0=new_sample, bit1=echo, bit2=trig
    // 0x4 last_echo_cnt
    // 0x8 current_echo_cnt
    // 0xC sample_id
    // ------------------------------
    always @(*) begin
        case (s0_address)
            2'd0: s0_readdata = {29'd0, trig, echo_ff2, new_sample};
            2'd1: s0_readdata = last_echo_cnt;
            2'd2: s0_readdata = echo_cnt;
            2'd3: s0_readdata = sample_id;
            default: s0_readdata = 32'd0;
        endcase
    end

endmodule
```

Listing 7: `rtl/ultra_sensor_csr.v`: complete HC-SR04 trigger/echo CSR.

```
module vga_counter (
    input  logic        clk50,
    reset,
    enable,
    output logic [10:0] hcount,  // hcount[10:1] is pixel column
    output logic [ 9:0] vcount,  // vcount[9:0] is pixel row
    output logic        VGA_CLK,
    VGA_HS,
    VGA_VS,
    VGA_BLANK_N,
    VGA_SYNC_N,
    endOfField
);

    /*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
```

```
 * HCOUNT 1599 0             1279       1599 0
 *              _______________              ________
 * ____________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _____________________     _____________
 * |____|       VGA_HS          |___|
 */
    // Parameters for hcount
    parameter   HACTIVE      = 11'd 1280,
                HFRONT_PORCH = 11'd 32,
                HSYNC        = 11'd 192,
                HBACK_PORCH  = 11'd 96,
                HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                               HBACK_PORCH; // 1600

    // Parameters for vcount
    parameter   VACTIVE      = 10'd 480,
                VFRONT_PORCH = 10'd 10,
                VSYNC        = 10'd 2,
                VBACK_PORCH  = 10'd 33,
                VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                               VBACK_PORCH; // 525

    logic endOfLine;

    always_ff @(posedge clk50 or posedge reset)
        if (reset) hcount <= 0;
        else if (enable) begin  // enable counting ---------------------------------- [Change]
            if (endOfLine) hcount <= 0;
            else hcount <= hcount + 11'd1;
        end

    assign endOfLine = hcount == HTOTAL - 1;

    //logic endOfField;

    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;

    // Horizontal sync: from 0x520 to 0x5DF (0x57F)
    // 101 0010 0000 to 101 1101 1111
    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;  // For putting sync on the green signal; unused

    // Horizontal active: 0 to 1279     Vertical active: 0 to 479
    // 101 0000 0000   -  1280          01 1110 0000  -  480
    // 110 0011 1111   -  1599          10 0000 1100  -  524
    assign VGA_BLANK_N = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                !( vcount[9] | (vcount[8:5] == 4'b1111) );

    /* VGA_CLK is 25 MHz
    *             __    __    __
    * clk50    __|  |__|  |__|
    *
    *           _____       __
    * hcount[0]__|     |_____|
    */
    assign VGA_CLK = hcount[0];  // 25 MHz clock: rising edge sensitive

```

```
endmodule
```

Listing 8: `rtl/vga_counter.sv`: complete VGA timing generator.

```
// 8-bit angle to Q1.10 sine/cosine, using a quarter-wave LUT.
//  [ θcos   θsin ]
//  [-θsin   θcos ]
module rot_lut (
    input  logic [7:0] angle,
    output logic signed [11:0] cos_q10,
    output logic signed [11:0] sin_q10
);

    logic signed [11:0] sin_b;
    logic signed [11:0] cos_b;
    logic [6:0]         phase;

    function automatic logic signed [11:0] sin_base(input logic [6:0] idx);
        begin
            unique case (idx)
            7'd0: sin_base = 12'sd0;
            7'd1: sin_base = 12'sd25;
            7'd2: sin_base = 12'sd50;
            7'd3: sin_base = 12'sd75;
            7'd4: sin_base = 12'sd100;
            7'd5: sin_base = 12'sd125;
            7'd6: sin_base = 12'sd150;
            7'd7: sin_base = 12'sd175;
            7'd8: sin_base = 12'sd200;
            7'd9: sin_base = 12'sd224;
            7'd10: sin_base = 12'sd249;
            7'd11: sin_base = 12'sd273;
            7'd12: sin_base = 12'sd297;
            7'd13: sin_base = 12'sd321;
            7'd14: sin_base = 12'sd345;
            7'd15: sin_base = 12'sd369;
            7'd16: sin_base = 12'sd392;
            7'd17: sin_base = 12'sd415;
            7'd18: sin_base = 12'sd438;
            7'd19: sin_base = 12'sd460;
            7'd20: sin_base = 12'sd483;
            7'd21: sin_base = 12'sd505;
            7'd22: sin_base = 12'sd526;
            7'd23: sin_base = 12'sd548;
            7'd24: sin_base = 12'sd569;
            7'd25: sin_base = 12'sd590;
            7'd26: sin_base = 12'sd610;
            7'd27: sin_base = 12'sd630;
            7'd28: sin_base = 12'sd650;
            7'd29: sin_base = 12'sd669;
            7'd30: sin_base = 12'sd688;
            7'd31: sin_base = 12'sd706;
            7'd32: sin_base = 12'sd724;
            7'd33: sin_base = 12'sd742;
            7'd34: sin_base = 12'sd759;
            7'd35: sin_base = 12'sd775;
            7'd36: sin_base = 12'sd792;
            7'd37: sin_base = 12'sd807;
            7'd38: sin_base = 12'sd822;
            7'd39: sin_base = 12'sd837;
            7'd40: sin_base = 12'sd851;
            7'd41: sin_base = 12'sd865;
            7'd42: sin_base = 12'sd878;
            7'd43: sin_base = 12'sd891;
            7'd44: sin_base = 12'sd903;
            7'd45: sin_base = 12'sd915;
            7'd46: sin_base = 12'sd926;
            7'd47: sin_base = 12'sd936;
```

```
            7'd48: sin_base = 12'sd946;
            7'd49: sin_base = 12'sd955;
            7'd50: sin_base = 12'sd964;
            7'd51: sin_base = 12'sd972;
            7'd52: sin_base = 12'sd980;
            7'd53: sin_base = 12'sd987;
            7'd54: sin_base = 12'sd993;
            7'd55: sin_base = 12'sd999;
            7'd56: sin_base = 12'sd1004;
            7'd57: sin_base = 12'sd1009;
            7'd58: sin_base = 12'sd1013;
            7'd59: sin_base = 12'sd1016;
            7'd60: sin_base = 12'sd1019;
            7'd61: sin_base = 12'sd1021;
            7'd62: sin_base = 12'sd1023;
            7'd63: sin_base = 12'sd1024;
            7'd64: sin_base = 12'sd1024;
            default: sin_base = 12'sd0;
            endcase
        end
    endfunction

    always_comb begin
        phase = {1'b0, angle[5:0]};
        sin_b = sin_base(phase);
        cos_b = sin_base(7'd64 - phase);

        unique case (angle[7:6])
        2'd0: begin
            cos_q10 =  cos_b;
            sin_q10 =  sin_b;
        end
        2'd1: begin
            cos_q10 = -sin_b;
            sin_q10 =  cos_b;
        end
        2'd2: begin
            cos_q10 = -cos_b;
            sin_q10 = -sin_b;
        end
        default: begin
            cos_q10 =  sin_b;
            sin_q10 = -cos_b;
        end
        endcase
    end

endmodule
```

Listing 9: `rtl/rot_lut.sv`: complete fixed-point sine/cosine lookup table.

```
// 64x64 RGBA4444 sprite ROM, initialized from one $readmemh file.
module sprite_rom #(
    parameter INIT_FILE = ""
) (
    input  logic        clk,
    input  logic [11:0] addr,
    output logic [15:0] pixel
);

    (* ramstyle = "M10K" *) logic [15:0] mem [0:4095];

    initial begin
        if (INIT_FILE != "")
            $readmemh(INIT_FILE, mem);
    end

    always_ff @(posedge clk) begin
```

```
        pixel <= mem[addr];
    end

endmodule
```

Listing 10: `rtl/sprite_rom.sv`: complete 64 by 64 RGBA4444 sprite ROM wrapper.

```
// 32x32 RGB888 tile ROM, initialized from one $readmemh file.
module tile_rom_rgb888 #(
    parameter INIT_FILE = ""
) (
    input  logic        clk,
    input  logic [9:0]  addr,
    output logic [23:0] pixel
);

    (* ramstyle = "M10K" *) logic [23:0] mem [0:1023];

    initial begin
        if (INIT_FILE != "")
            $readmemh(INIT_FILE, mem);
    end

    always_ff @(posedge clk) begin
        pixel <= mem[addr];
    end

endmodule
```

Listing 11: `rtl/tile_rom_rgb888.sv`: complete 32 by 32 RGB888 tile ROM wrapper.

B Appendix: Selected DE1-SoC GHRD Integration Code

These listings are selected because they are the handoff points between the custom project logic and the DE1-SoC base system: the board wrapper, the Platform Designer component definition, the generated device-tree node used by the Linux driver, the Qsys connection from the HPS lightweight bridge, and the relevant pin assignments. Large generated descriptions such as the full `soc_system.sopcinfo.xml` are not included because they are not useful for understanding the custom design.

B.1 Board-Level Wrapper Connections

```
// de1_soc_top.v
// Quartus top-level wrapper for Fruit Ninja DE1-SoC  (ultrasonic version)
//
// FPGA-side port names match Terasic DE1_SoC_pin_assignments.qsf
// HPS-side port names kept from GHRD (no QSF pin assignment needed for HPS)
// Changes vs GHRD: TFT -> VGA DAC, HC-SR04 on GPIO_0[0]/GPIO_1[0]

module de1_soc_top (

    // ==================== FPGA Clock ====================
    input  wire        CLOCK_50,              // PIN_AF14

    // ==================== KEY (active-low pushbuttons) ====================
    input  wire [1:0]  KEY,                   // KEY[0]=PIN_AA14, KEY[1]=PIN_AA15

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

    // ==================== VGA (ADV7123 DAC) ====================
    output wire [7:0]  VGA_R,
```

```
    output wire [7:0]  VGA_G,
    output wire [7:0]  VGA_B,
    output wire        VGA_CLK,
    output wire        VGA_HS,
    output wire        VGA_VS,
    output wire        VGA_BLANK_N,
    output wire        VGA_SYNC_N,

    // ==================== GPIO_0 (HC-SR04 echo on [0]) ====================
    inout  wire [35:0] GPIO_0,

    // ==================== GPIO_1 (HC-SR04 trig on [0]) ====================
    inout  wire [35:0] GPIO_1,

    // ==================== HPS DDR3 ====================
```

Listing 12: DE1_SoC_GHRD/de1_soc_top.v: project-relevant top-level ports for VGA and ultrasonic GPIO.

```
    wire clk_66m;

    // PLL: 50 MHz -> 66 MHz (kept from GHRD, still referenced by soc_system)
    PLL_0002 pll_inst (
        .refclk   (CLOCK_50),
        .rst      (1'b0),
        .outclk_0 (),         // 33 MHz - unused (was for TFT)
        .outclk_1 (clk_66m),  // 66 MHz - wired to soc_system.clk_66m_clk
        .locked   ()
    );

    wire [9:0] led_pio_unused;   // GHRD-built-in led_pio, not used
    wire [9:0] led_from_game;    // driven by top_module (LED[0]=trig, LED[1]=echo)
    wire       sensor_echo;
    wire       sensor_trig;

    assign sensor_echo = GPIO_0[0];
    assign GPIO_1[0]   = sensor_trig;

    assign LEDR = led_from_game;

    // ============================================================
    //  Qsys soc_system instantiation
    // ============================================================
    soc_system u0 (
        // ---- Clock & Reset ----
        .clk_clk                            (CLOCK_50),
        .reset_reset_n                      (hps_0_h2f_reset_n),
        .clk_66m_clk                        (clk_66m),
        .hps_0_h2f_reset_reset_n            (hps_0_h2f_reset_n),

        // ---- HPS DDR3 Memory ----
        .memory_mem_a                       (memory_mem_a),
        .memory_mem_ba                      (memory_mem_ba),
        .memory_mem_ck                      (memory_mem_ck),
        .memory_mem_ck_n                    (memory_mem_ck_n),
        .memory_mem_cke                     (memory_mem_cke),
        .memory_mem_cs_n                    (memory_mem_cs_n),
        .memory_mem_ras_n                   (memory_mem_ras_n),
        .memory_mem_cas_n                   (memory_mem_cas_n),
        .memory_mem_we_n                    (memory_mem_we_n),
        .memory_mem_reset_n                 (memory_mem_reset_n),
        .memory_mem_dq                      (memory_mem_dq),
        .memory_mem_dqs                     (memory_mem_dqs),
        .memory_mem_dqs_n                   (memory_mem_dqs_n),
        .memory_mem_odt                     (memory_mem_odt),
        .memory_mem_dm                      (memory_mem_dm),
        .memory_oct_rzqin                   (memory_oct_rzqin),
```

```
49
50          // ---- HPS Ethernet ----
51          .hps_0_hps_io_hps_io_emac1_inst_TX_CLK (hps_eth1_TX_CLK),
52          .hps_0_hps_io_hps_io_emac1_inst_TXD0   (hps_eth1_TXD0),
53          .hps_0_hps_io_hps_io_emac1_inst_TXD1   (hps_eth1_TXD1),
54          .hps_0_hps_io_hps_io_emac1_inst_TXD2   (hps_eth1_TXD2),
55          .hps_0_hps_io_hps_io_emac1_inst_TXD3   (hps_eth1_TXD3),
56          .hps_0_hps_io_hps_io_emac1_inst_RXD0   (hps_eth1_RXD0),
57          .hps_0_hps_io_hps_io_emac1_inst_MDIO   (hps_eth1_MDIO),
58          .hps_0_hps_io_hps_io_emac1_inst_MDC    (hps_eth1_MDC),
59          .hps_0_hps_io_hps_io_emac1_inst_RX_CTL (hps_eth1_RX_CTL),
60          .hps_0_hps_io_hps_io_emac1_inst_TX_CTL (hps_eth1_TX_CTL),
61          .hps_0_hps_io_hps_io_emac1_inst_RX_CLK (hps_eth1_RX_CLK),
62          .hps_0_hps_io_hps_io_emac1_inst_RXD1   (hps_eth1_RXD1),
63          .hps_0_hps_io_hps_io_emac1_inst_RXD2   (hps_eth1_RXD2),
64          .hps_0_hps_io_hps_io_emac1_inst_RXD3   (hps_eth1_RXD3),
65
66          // ---- HPS SD Card ----
67          .hps_0_hps_io_hps_io_sdio_inst_CMD     (hps_sdio_CMD),
68          .hps_0_hps_io_hps_io_sdio_inst_D0      (hps_sdio_D0),
69          .hps_0_hps_io_hps_io_sdio_inst_D1      (hps_sdio_D1),
70          .hps_0_hps_io_hps_io_sdio_inst_CLK     (hps_sdio_CLK),
71          .hps_0_hps_io_hps_io_sdio_inst_D2      (hps_sdio_D2),
72          .hps_0_hps_io_hps_io_sdio_inst_D3      (hps_sdio_D3),
73
74          // ---- HPS USB ----
75          .hps_0_hps_io_hps_io_usb1_inst_D0      (hps_usb1_D0),
76          .hps_0_hps_io_hps_io_usb1_inst_D1      (hps_usb1_D1),
77          .hps_0_hps_io_hps_io_usb1_inst_D2      (hps_usb1_D2),
78          .hps_0_hps_io_hps_io_usb1_inst_D3      (hps_usb1_D3),
79          .hps_0_hps_io_hps_io_usb1_inst_D4      (hps_usb1_D4),
80          .hps_0_hps_io_hps_io_usb1_inst_D5      (hps_usb1_D5),
81          .hps_0_hps_io_hps_io_usb1_inst_D6      (hps_usb1_D6),
82          .hps_0_hps_io_hps_io_usb1_inst_D7      (hps_usb1_D7),
83          .hps_0_hps_io_hps_io_usb1_inst_CLK     (hps_usb1_CLK),
84          .hps_0_hps_io_hps_io_usb1_inst_STP     (hps_usb1_STP),
85          .hps_0_hps_io_hps_io_usb1_inst_DIR     (hps_usb1_DIR),
86          .hps_0_hps_io_hps_io_usb1_inst_NXT     (hps_usb1_NXT),
87
88          // ---- HPS SPI ----
89          .hps_0_hps_io_hps_io_spim0_inst_CLK    (hps_spim0_CLK),
90          .hps_0_hps_io_hps_io_spim0_inst_MOSI   (hps_spim0_MOSI),
91          .hps_0_hps_io_hps_io_spim0_inst_MISO   (hps_spim0_MISO),
92          .hps_0_hps_io_hps_io_spim0_inst_SS0    (hps_spim0_SS0),
93          .hps_0_hps_io_hps_io_spim1_inst_CLK    (hps_spim1_CLK),
94          .hps_0_hps_io_hps_io_spim1_inst_MOSI   (hps_spim1_MOSI),
95          .hps_0_hps_io_hps_io_spim1_inst_MISO   (hps_spim1_MISO),
96          .hps_0_hps_io_hps_io_spim1_inst_SS0    (hps_spim1_SS0),
97
98          // ---- HPS UART ----
99          .hps_0_hps_io_hps_io_uart0_inst_RX     (hps_uart0_RX),
100         .hps_0_hps_io_hps_io_uart0_inst_TX     (hps_uart0_TX),
101
102         // ---- HPS I2C ----
103         .hps_0_hps_io_hps_io_i2c0_inst_SDA     (hps_i2c0_SDA),
104         .hps_0_hps_io_hps_io_i2c0_inst_SCL     (hps_i2c0_SCL),
105         .hps_0_hps_io_hps_io_i2c1_inst_SDA     (hps_i2c1_SDA),
106         .hps_0_hps_io_hps_io_i2c1_inst_SCL     (hps_i2c1_SCL),
107
108         // ---- HPS GPIO ----
109         .hps_0_hps_io_hps_io_gpio_inst_GPIO00  (hps_gpio_GPIO00),
110         .hps_0_hps_io_hps_io_gpio_inst_GPIO09  (hps_usb1_CONV_N),
111         .hps_0_hps_io_hps_io_gpio_inst_GPIO34  (hps_gpio_GPIO34),
112         .hps_0_hps_io_hps_io_gpio_inst_GPIO35  (hps_eth1_INT_N),
113         .hps_0_hps_io_hps_io_gpio_inst_GPIO37  (hps_gpio_GPIO37),
114         .hps_0_hps_io_hps_io_gpio_inst_GPIO44  (hps_gpio_GPIO44),
115         .hps_0_hps_io_hps_io_gpio_inst_GPIO48  (hps_gpio_GPIO48),
116         .hps_0_hps_io_hps_io_gpio_inst_GPIO53  (hps_led),
```

```
        .hps_0_hps_io_hps_io_gpio_inst_GPIO54  (hps_key),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO61  (hps_gpio_GPIO61),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO62  (hps_gpio_GPIO62),

        // ---- Button PIO ----
        .button_pio_export                     (KEY),

        // ---- GHRD LED PIO (unused, top_module drives LEDR via led conduit) ----
        .led_pio_export                        (led_pio_unused),

        // ---- FPGA SPI/UART kept but unused ----
        .spi_0_MISO                            (1'b0),
        .spi_0_MOSI                            (),
        .spi_0_SCLK                            (),
        .spi_0_SS_n                            (),
        .uart_0_rxd                            (1'b1),
        .uart_0_txd                            (),
        .uart_1_rxd                            (1'b1),
        .uart_1_txd                            (),

        // ---- Ultrasonic conduit (echo=GPIO_0[0], trig=GPIO_1[0]) ----
        .ultra_echo_in                         (sensor_echo),
        .ultra_trig_out                        (sensor_trig),

        // ---- VGA conduit ----
        .vga_vga_r                             (VGA_R),
        .vga_vga_g                             (VGA_G),
        .vga_vga_b                             (VGA_B),
        .vga_vga_clk                           (VGA_CLK),
        .vga_vga_hs                            (VGA_HS),
        .vga_vga_vs                            (VGA_VS),
        .vga_vga_blank_n                       (VGA_BLANK_N),
        .vga_vga_sync_n                        (VGA_SYNC_N),

        // ---- Game LED conduit (10-bit from top_module) ----
        .led_led                               (led_from_game)
    );
```

Listing 13: `DE1_SoC_GHRD/de1_soc_top.v`: custom conduit wiring for sensor, VGA, LEDs, and the generated `soc_system`.

B.2 Platform Designer Custom Component

```
# module top_module
#
set_module_property DESCRIPTION "Fruit Ninja game peripheral"
set_module_property NAME top_module
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP Team3
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME "Fruit Ninja Top Module"
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false


#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL top_module
```

```
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file top_module.sv SYSTEM_VERILOG PATH ../rtl/top_module.sv TOP_LEVEL_FILE
add_fileset_file sprite_rom.sv SYSTEM_VERILOG PATH ../rtl/sprite_rom.sv
add_fileset_file tile_rom_rgb888.sv SYSTEM_VERILOG PATH ../rtl/tile_rom_rgb888.sv
add_fileset_file rot_lut.sv SYSTEM_VERILOG PATH ../rtl/rot_lut.sv
add_fileset_file ultra_sensor_csr.v VERILOG PATH ../rtl/ultra_sensor_csr.v
add_fileset_file vga_counter.sv SYSTEM_VERILOG PATH ../rtl/vga_counter.sv
add_fileset_file apple.hex OTHER PATH ../pics/apple.hex
add_fileset_file apple_blow.hex OTHER PATH ../pics/apple_blow.hex
add_fileset_file banana.hex OTHER PATH ../pics/banana.hex
add_fileset_file banana_blow.hex OTHER PATH ../pics/banana_blow.hex
add_fileset_file watermelon.hex OTHER PATH ../pics/watermelon.hex
add_fileset_file watermelon_blow.hex OTHER PATH ../pics/watermelon_blow.hex
add_fileset_file bomb.hex OTHER PATH ../pics/bomb.hex
add_fileset_file boom_blow.hex OTHER PATH ../pics/boom_blow.hex
add_fileset_file tile_0000.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0000.hex
add_fileset_file tile_0001.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0001.hex
add_fileset_file tile_0002.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0002.hex
add_fileset_file tile_0003.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0003.hex
add_fileset_file tile_0004.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0004.hex
add_fileset_file tile_0005.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0005.hex
add_fileset_file tile_0006.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0006.hex
add_fileset_file tile_0007.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0007.hex
add_fileset_file tile_0008.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0008.hex
add_fileset_file tile_0009.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0009.hex
add_fileset_file tile_0010.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0010.hex
add_fileset_file tile_0011.hex OTHER PATH ../pics/tiles_32_rgb888_hex/0011.hex
```

Listing 14: `DE1_SoC_GHRD/top_module_hw.tcl`: custom component metadata and source/asset file set.

```
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset reset reset Input 1


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

add_interface_port avalon_slave chipselect chipselect Input 1
add_interface_port avalon_slave address address Input 4
```

```
add_interface_port avalon_slave read read Input 1
add_interface_port avalon_slave readdata readdata Output 32
add_interface_port avalon_slave write write Input 1
add_interface_port avalon_slave writedata writedata Input 32
set_interface_assignment avalon_slave embeddedsw.configuration.isFlash 0
set_interface_assignment avalon_slave embeddedsw.configuration.isMemoryDevice 0
set_interface_assignment avalon_slave embeddedsw.configuration.isNonVolatileStorage 0
set_interface_assignment avalon_slave embeddedsw.configuration.isPrintableDevice 0


#
# connection point vga
#
add_interface vga conduit end
set_interface_property vga associatedClock clock
set_interface_property vga associatedReset ""
set_interface_property vga ENABLED true
set_interface_property vga EXPORT_OF ""
set_interface_property vga PORT_NAME_MAP ""
set_interface_property vga CMSIS_SVD_VARIABLES ""
set_interface_property vga SVD_ADDRESS_GROUP ""

add_interface_port vga VGA_R vga_r Output 8
add_interface_port vga VGA_G vga_g Output 8
add_interface_port vga VGA_B vga_b Output 8
add_interface_port vga VGA_CLK vga_clk Output 1
add_interface_port vga VGA_HS vga_hs Output 1
add_interface_port vga VGA_VS vga_vs Output 1
add_interface_port vga VGA_BLANK_N vga_blank_n Output 1
add_interface_port vga VGA_SYNC_N vga_sync_n Output 1


#
# connection point ultra
#
add_interface ultra conduit end
set_interface_property ultra associatedClock clock
set_interface_property ultra associatedReset ""
set_interface_property ultra ENABLED true
set_interface_property ultra EXPORT_OF ""
set_interface_property ultra PORT_NAME_MAP ""
set_interface_property ultra CMSIS_SVD_VARIABLES ""
set_interface_property ultra SVD_ADDRESS_GROUP ""

add_interface_port ultra echo_in  echo_in  Input  1
add_interface_port ultra trig_out trig_out Output 1


#
# connection point LED
#
add_interface LED conduit end
set_interface_property LED associatedClock clock
set_interface_property LED associatedReset ""
set_interface_property LED ENABLED true
set_interface_property LED EXPORT_OF ""
set_interface_property LED PORT_NAME_MAP ""
set_interface_property LED CMSIS_SVD_VARIABLES ""
set_interface_property LED SVD_ADDRESS_GROUP ""

add_interface_port LED LED led Output 10
```

Listing 15: `DE1_SoC_GHRD/top_module_hw.tcl`: Avalon, VGA, ultrasonic, and LED interface declarations.

B.3 Device Tree and Qsys Address Connection

```
        #size-cells = <1>;
        compatible = "ALTR,avalon", "simple-bus";
        bus-frequency = <0>;

        hps_0_bridges: bridge@0xc0000000 {
            compatible = "altr,bridge-21.1", "simple-bus";
            reg = <0xc0000000 0x20000000>,
                <0xff200000 0x00200000>;
            reg-names = "axi_h2f", "axi_h2f_lw";
            clocks = <&clk_0 &clk_0>;
            clock-names = "h2f_axi_clock", "h2f_lw_axi_clock";
            #address-cells = <2>;
            #size-cells = <1>;
            ranges = <0x00000001 0x00000000 0xff200000 0x00000040>,
                <0x00000001 0x000000e0 0xff2000e0 0x00000008>,
                <0x00000001 0x000000d0 0xff2000d0 0x00000010>,
                <0x00000001 0x000000c0 0xff2000c0 0x00000010>,
                <0x00000001 0x000000a0 0xff2000a0 0x00000020>,
                <0x00000001 0x00000040 0xff200040 0x00000020>,
                <0x00000001 0x00000080 0xff200080 0x00000020>;

            top_module_0: unknown@0x100000000 {
                compatible = "csee4840,top_module-1.0";
                reg = <0x00000001 0x00000000 0x00000040>;
                clocks = <&clk_0>;
            }; //end unknown@0x100000000 (top_module_0)
```

Listing 16: `DE1_SoC_GHRD/soc_system.dts`: bridge range and `top_module` device-tree node used by the Linux driver.

```
 <module name="top_module_0" kind="top_module" version="1.0" enabled="1" />
 <module name="uart_0" kind="altera_avalon_uart" version="21.1" enabled="1">
  <parameter name="baud" value="115200" />
  <parameter name="clockRate" value="50000000" />
  <parameter name="dataBits" value="8" />
  <parameter name="fixedBaud" value="false" />
  <parameter name="parity" value="NONE" />
  <parameter name="simCharStream" value="" />
  <parameter name="simInteractiveInputEnable" value="false" />
  <parameter name="simInteractiveOutputEnable" value="false" />
  <parameter name="simTrueBaud" value="false" />
  <parameter name="stopBits" value="1" />
  <parameter name="syncRegDepth" value="2" />
  <parameter name="useCtsRts" value="false" />
  <parameter name="useEopRegister" value="false" />
  <parameter name="useRelativePathForSimFile" value="false" />
 </module>
 <module name="uart_1" kind="altera_avalon_uart" version="21.1" enabled="1">
  <parameter name="baud" value="115200" />
  <parameter name="clockRate" value="50000000" />
  <parameter name="dataBits" value="8" />
  <parameter name="fixedBaud" value="true" />
  <parameter name="parity" value="NONE" />
  <parameter name="simCharStream" value="" />
  <parameter name="simInteractiveInputEnable" value="false" />
  <parameter name="simInteractiveOutputEnable" value="false" />
  <parameter name="simTrueBaud" value="false" />
  <parameter name="stopBits" value="1" />
  <parameter name="syncRegDepth" value="2" />
  <parameter name="useCtsRts" value="false" />
  <parameter name="useEopRegister" value="false" />
  <parameter name="useRelativePathForSimFile" value="false" />
 </module>
 <connection
   kind="avalon"
```

```
   version="21.1"
   start="hps_0.h2f_lw_axi_master"
   end="top_module_0.avalon_slave">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x0000" />
  <parameter name="defaultConnection" value="false" />
 </connection>
 <connection
```

Listing 17: `DE1_SoC_GHRD/soc_system.qsys`: `top_module_0` component and lightweight HPS-to-FPGA bridge connection.

B.4 Relevant Pin Assignment Excerpts

```
set_location_assignment PIN_AF18 -to GPIO_0[32]
set_location_assignment PIN_AG20 -to GPIO_0[33]
set_location_assignment PIN_AG18 -to GPIO_0[34]
set_location_assignment PIN_AJ21 -to GPIO_0[35]
set_location_assignment PIN_Y18 -to GPIO_0[3]
set_location_assignment PIN_AK16 -to GPIO_0[4]
set_location_assignment PIN_AK18 -to GPIO_0[5]
set_location_assignment PIN_AK19 -to GPIO_0[6]
set_location_assignment PIN_AJ19 -to GPIO_0[7]
set_location_assignment PIN_AJ17 -to GPIO_0[8]
set_location_assignment PIN_AJ16 -to GPIO_0[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[0]
```

Listing 18: `DE1_SoC_GHRD/DE1_SoC_pin_assignments.qsf`: GPIO_0 assignments including sensor echo input.

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

```
set_location_assignment PIN_AF26 -to GPIO_1[8]
set_location_assignment PIN_AG25 -to GPIO_1[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[0]
```

Listing 19: `DE1_SoC_GHRD/DE1_SoC_pin_assignments.qsf`: GPIO_1 assignments including sensor trigger output.

```
# LEDR
#============================================================
set_location_assignment PIN_V16 -to LEDR[0]
set_location_assignment PIN_W16 -to LEDR[1]
set_location_assignment PIN_V17 -to LEDR[2]
set_location_assignment PIN_V18 -to LEDR[3]
set_location_assignment PIN_W17 -to LEDR[4]
set_location_assignment PIN_W19 -to LEDR[5]
set_location_assignment PIN_Y19 -to LEDR[6]
set_location_assignment PIN_W20 -to LEDR[7]
set_location_assignment PIN_W21 -to LEDR[8]
set_location_assignment PIN_Y21 -to LEDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[9]
```

Listing 20: `DE1_SoC_GHRD/DE1_SoC_pin_assignments.qsf`: LEDR pin assignments used for debug output.

```
set_location_assignment PIN_B13 -to VGA_B[0]
set_location_assignment PIN_G13 -to VGA_B[1]
set_location_assignment PIN_H13 -to VGA_B[2]
set_location_assignment PIN_F14 -to VGA_B[3]
set_location_assignment PIN_H14 -to VGA_B[4]
set_location_assignment PIN_F15 -to VGA_B[5]
set_location_assignment PIN_G15 -to VGA_B[6]
set_location_assignment PIN_J14 -to VGA_B[7]
set_location_assignment PIN_F10 -to VGA_BLANK_N
set_location_assignment PIN_A11 -to VGA_CLK
set_location_assignment PIN_J9 -to VGA_G[0]
set_location_assignment PIN_J10 -to VGA_G[1]
set_location_assignment PIN_H12 -to VGA_G[2]
set_location_assignment PIN_G10 -to VGA_G[3]
set_location_assignment PIN_G11 -to VGA_G[4]
set_location_assignment PIN_G12 -to VGA_G[5]
set_location_assignment PIN_F11 -to VGA_G[6]
set_location_assignment PIN_E11 -to VGA_G[7]
set_location_assignment PIN_B11 -to VGA_HS
set_location_assignment PIN_A13 -to VGA_R[0]
set_location_assignment PIN_C13 -to VGA_R[1]
set_location_assignment PIN_E13 -to VGA_R[2]
set_location_assignment PIN_B12 -to VGA_R[3]
set_location_assignment PIN_C12 -to VGA_R[4]
set_location_assignment PIN_D12 -to VGA_R[5]
set_location_assignment PIN_E12 -to VGA_R[6]
set_location_assignment PIN_F13 -to VGA_R[7]
set_location_assignment PIN_C10 -to VGA_SYNC_N
set_location_assignment PIN_D11 -to VGA_VS
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[3]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_BLANK_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_HS
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_SYNC_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_VS
```

Listing 21: `DE1_SoC_GHRD/DE1_SoC_pin_assignments.qsf`: VGA DAC pin assignments.