# Ultrasonic Fruit Ninja on DE1-SoC

## Design Document

**CSEE 4840 Embedded System Design — Spring 2026**

**Team 3**: Peiheng Li(pl2978), Chengrui Li(cl4750), Yitong Bai(yb2636)

**Device**: Terasic DE1-SoC (Cyclone V SE 5CSEMA5F31C6), HC-SR04 Ultrasonic Sensor

# Contents

# 1 Introduction

We are building a *Fruit Ninja* game driven by an **HC-SR04 ultrasonic range finder** on the DE1-SoC. A VGA monitor shows a dashed horizontal line across the middle of the screen; an orange fruit is thrown from the bottom of the screen and follows a parabolic trajectory. The player waves a hand in front of the sensor: if the wave happens while the fruit is crossing the line, the fruit is cut and the score goes up; otherwise a life is lost. Three missed fruits ends the game.

The project replaces an earlier camera-based "motion detect" prototype because the camera pipeline was brittle under classroom lighting, burned large amounts of FPGA memory on a frame buffer, and required two clock domains (24 MHz PCLK + 50 MHz core). In order to make it on time, we turns to use ultrasonic sensor as the detection of cutting a fruit. A one-wire ultrasonic sensor gives one clean number, which is all we need to decide "is a hand moving in front of the board right now?".

The main design choices are listed below:

| **Decision** | **Choice** | **Why** |
|---|---|---|
| Sensing | HC-SR04 (40 kHz ultrasonic) | 1-D "is something near" is enough; trivial 2-pin interface |
| Video | FPGA 640$\times$480@60 Hz, combinational per-pixel geometry | No frame buffer needed; 1 fruit + 1 line + 1 score bar fit in $\sim$22-bit ALU logic |
| Game state | on HPS, Linux user space (`game`) | Easy to change threshold/physics without re-synthesis |

The overall system algorithm is as follows:

| **Timing** | **Action** |
|---|---|
| `every 100 ms` | FPGA fires a 100 $\mu$s trig pulse, times the echo return, latches W |
| `every 33 ms` | `game.c` reads (W, sample_id) via ioctl,<br>updates fruit (x, y) by parabolic physics,<br>if fruit crossed the line && a FAR to NEAR edge fired this frame $\rightarrow$ CUT (+1)<br>if fruit fell off the bottom without a cut $\rightarrow$ MISS ($-1$ life)<br>pushes (fruit_x, fruit_y, r, visible, score) back to FPGA registers |
| `every pixel` | FPGA renderer computes inside_fruit / on_line / in_score_bar and outputs RGB |

# 2 System Block Diagram

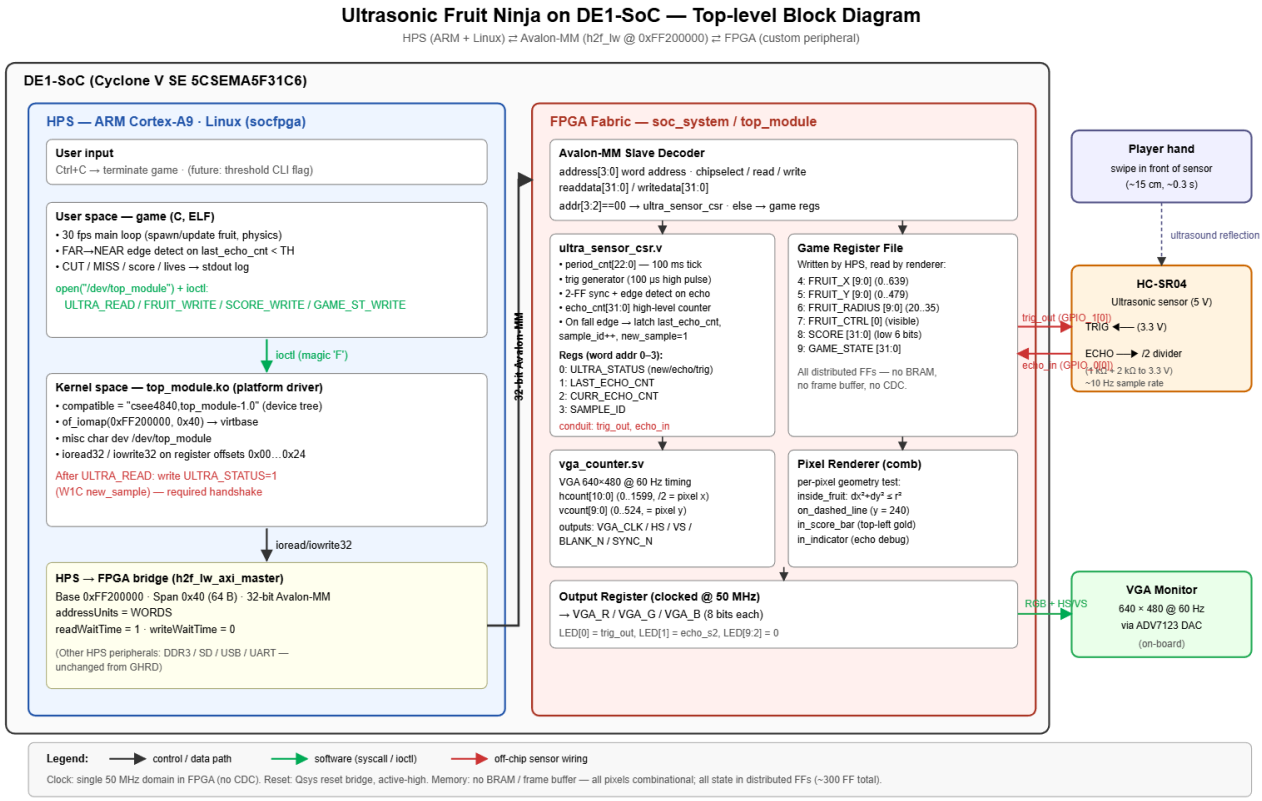


Figure 1: Top Level Block Diagram

The system has **ten functional blocks** split across three physical pieces (HPS, FPGA fabric, off-board sensor/display). We describe each block and then the communication pathway that connects it to its neighbours.

## 2.1 Functional blocks

| # | Block | Lives in | Purpose |
|---|---|---|---|
| B1 | user-space MMIO test program | HPS — user space | Reads ULTRA_STATUS / LAST_ECHO_CNT / SAMPLE_ID and prints the measured distance for bring-up validation |
| B2 | planned platform driver (future work) | HPS — kernel | Intended final software interface; not yet implemented in the current prototype |
| B3 | h2f_lw_axi_master | HPS hard IP | Lightweight HPS-to-FPGA bridge; 32-bit, base 0xFF200000, span 0x40 |
| B4 | Avalon-MM slave decoder | FPGA top_module.sv | 4-bit word-address decoder; routes addr[3:2]==00 to ultra_sensor_csr, else to the game register file |
| B5 | ultra_sensor_csr | FPGA | Generates trig, times echo, latches last_echo_cnt + sample_id, exposes status |
| B6 | Game register file | FPGA | 6 software-written registers: FRUIT_{X,Y,RADIUS,CTRL}, SCORE, GAME_STATE |
| B7 | vga_counter | FPGA | 640×480@60 Hz timing generator (hcount, vcount, HS/VS/BLANK/SYNC) |
| B8 | Pixel renderer | FPGA (comb.) | Per-pixel geometry test + priority mux → 24-bit RGB |
| B9 | ADV7123 DAC | On-board IC | Converts 24-bit digital RGB + sync to analog VGA |
| B10 | HC-SR04 | Off-board | Ultrasonic sensor; emits a burst on trig, returns echo whose high-time is proportional to range |

## 2.2 Communication pathways and their protocols

| Pathway | Carrier | Protocol / encoding | Width | Rate |
|---|---|---|---|---|
| game ↔ top_module.ko | syscall boundary | ioctl(fd, cmd, &arg) with magic 'F', copy_{from,to}_user for structs | struct size | ∼30 Hz (read) + ∼30 Hz (fruit write) |
| top_module.ko → h2f_lw_axi_master | kernel MMIO | ioread32 / iowrite32 on virtbase + off | 32 b | bursts of 1–4 transfers per ioctl |
| h2f_lw_axi_master ↔ top_module | AXI → Avalon-MM | Qsys bridge: addressUnits=WORDS, readWaitTime=1, writeWaitTime=0, no waitrequest | 32 b data, 4 b word addr | ≤ 100 transactions/s — negligible |
| Pixel renderer → ADV7123 | parallel RGB + sync | R/G/B each 8 b, HS/VS/BLANK_N/SYNC_N, pixel clock 25 MHz (hcount advances every 2 cycles of the 50 MHz clk) | 3×8 + 4 | 25 M pixel/s |
| FPGA ↔ HC-SR04 | GPIO | trig_out pushed to sensor TRIG; sensor ECHO (5 V) through 1 kΩ+2 kΩ divider to 3.3 V GPIO_0[0] | 1+1 | ∼10 Hz sample |
| FPGA internal buses | wires | single 50 MHz clock domain; active-high synchronous reset from Qsys reset bridge | see §5 | 50 MHz |

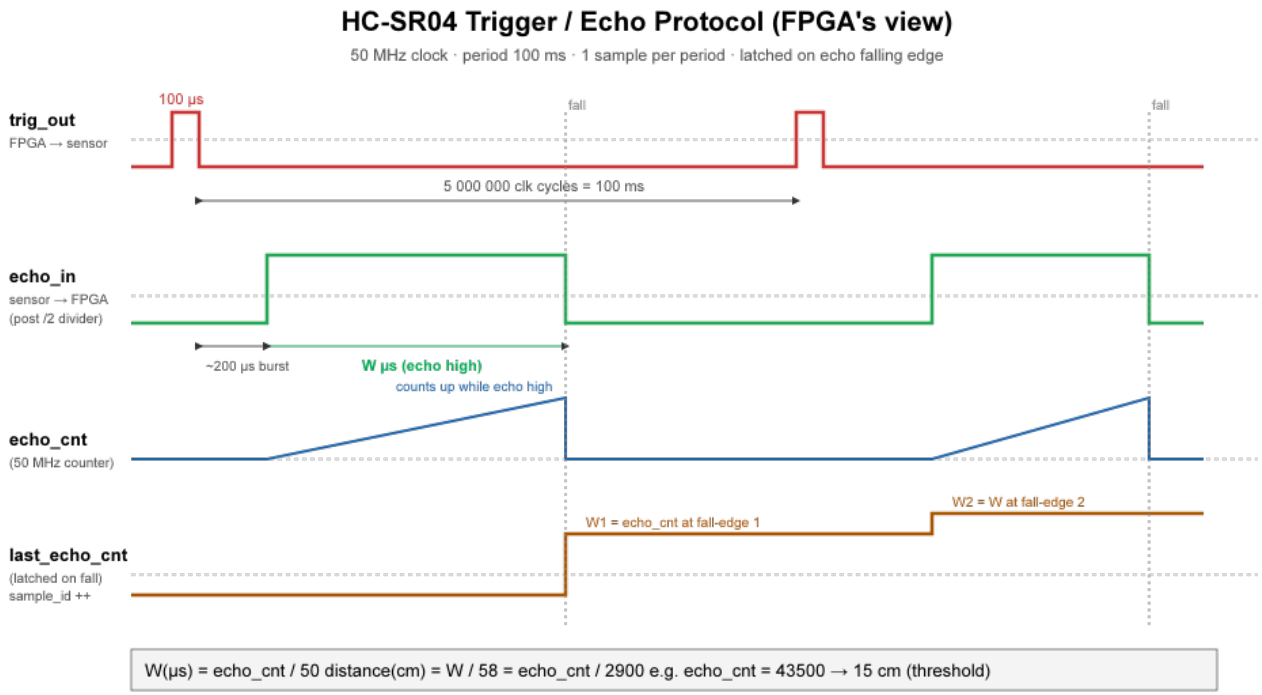


Figure 2: HC-SR04 protocol as seen by the FPGA.

# 3 Algorithms

## 3.1 HC-SR04 distance measurement

A simple trigger/echo ultrasonic sensor gives one clean number — the echo pulse width — which is all we need to decide whether a hand is in front of the board right now. The FPGA drives a 100 $\mu$s pulse on `TRIG`; the sensor emits an 8-cycle 40 kHz burst; ECHO goes high from the moment the burst is transmitted until its reflection returns. The high-time `W` is proportional to round-trip distance (`distance_cm = W_µs / 58`).

The FPGA implements this in `ultra_sensor_csr.v` as four cooperating blocks, all in the single 50 MHz domain:

1. **Trigger generator** — a 23-bit period counter wrapping every 5 000 000 cycles (100 ms); `trig_out` is asserted for about 100 $\mu$s once per measurement period. 100 ms gives plenty of head-room above the sensor's $\sim$24 ms max echo return.

2. **Echo synchroniser** — two flip-flops on `echo_in` plus one extra stage to derive rising/falling edges safely across the async boundary.

3. **Width counter** — a 32-bit counter cleared each period and incremented while the synchronised echo is high.

4. **Result latch** — on the echo falling edge, `last_echo_cnt` captures the counter value, `sample_id` increments by 1, and the `new_sample` dirty bit is asserted. `new_sample` is cleared only when software writes `0x1` back to `ULTRA_STATUS` (W1C handshake).

Using `sample_id` as the freshness signal (rather than `new_sample` alone) means software never mistakes a stale reading for a fresh one even if it reads twice between measurements.

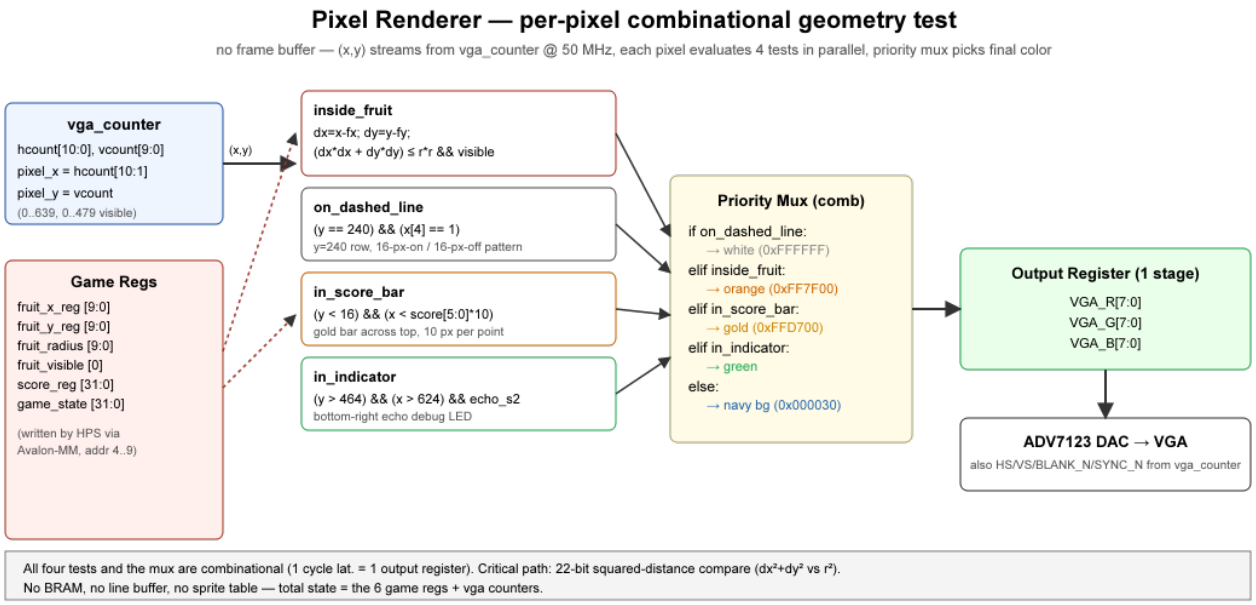


Figure 3: Combinational pixel renderer.

## 3.2 VGA pixel rendering

There is no frame buffer. Each pixel is computed on the fly from the current `(pixel_x, pixel_y)` coming out of `vga_counter` and the six game registers. Four geometry tests run in parallel:

- **inside_fruit** — $(x - fx)^2 + (y - fy)^2 \leq r^2$, gated by `visible`.
- **on_dashed_line** — the row $y = 240$, with alternating on/off groups of pixels to produce the dashed pattern.
- **in_score_bar** — a top-of-screen strip whose length is proportional to `score`.
- **in_indicator** — a small box in one corner driven by the live echo bit, purely for debugging.

A priority mux picks the winning color (dashed line $>$ fruit $>$ score bar $>$ indicator $>$ background) and a single output register stage drives the ADV7123 DAC. The critical path is the 22-bit squared-distance comparator; at 50 MHz this closes on the Cyclone V SE with comfortable slack, so no pipelining is needed.

## 3.3 Swipe detection

"The player swiped" is a software concept — the FPGA only reports a distance. We model it as a two-state machine (FAR / NEAR) driven by whether the most recent `last_echo_cnt` is below the threshold TH ($\sim$15 cm, expressed in counter units as $2900 \times 15 = 43500$). A swipe event is emitted **only** on the FAR $\rightarrow$ NEAR transition, never on "still near", so holding a hand in front of the sensor does not retrigger.

Two checks prevent false retriggering:

- **Freshness gate** — the state is updated only when `sample_id` has changed since the last read, so the same hardware sample is never consumed twice.
- **One-frame lifetime** — a pending swipe is consumed the frame the fruit centre crosses the line, otherwise discarded. Carrying it over would make the game feel laggy.

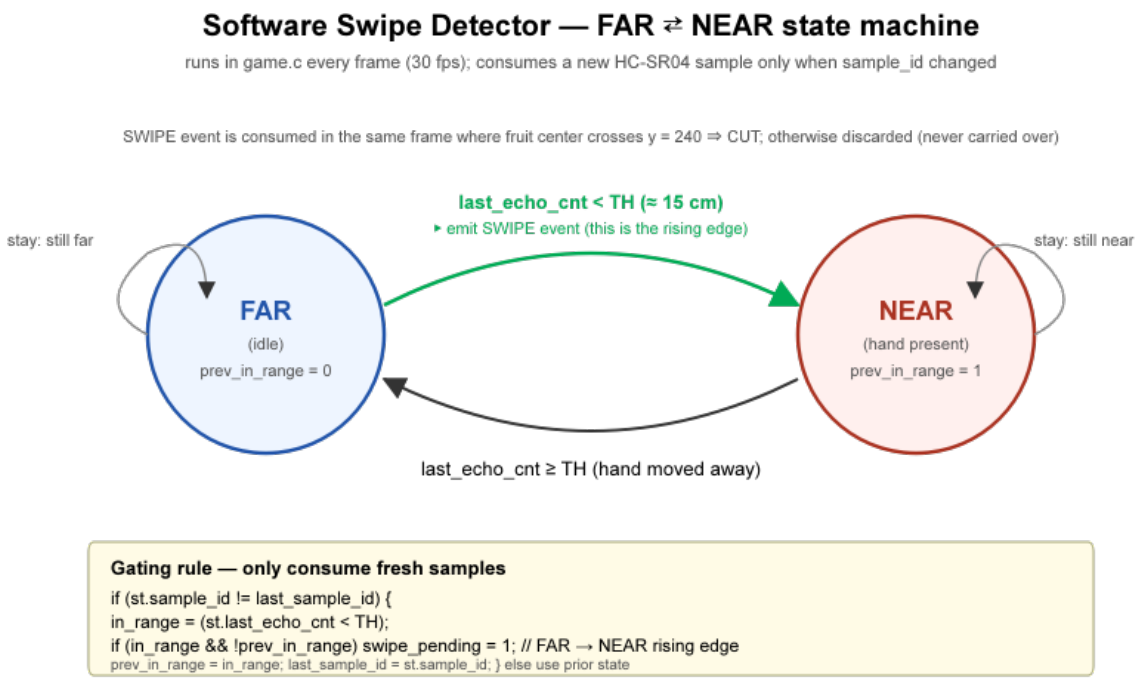


Figure 4: FAR ⇄ NEAR state machine in `game.c`.

### 3.4 Game logic (software)

The rest of `game.c` is standard 2D arcade bookkeeping:

- **Fruit physics** — each fruit has `(x, y, vx, vy)` in a fixed-point sub-pixel unit (integer math is enough at 30 fps; the sub-pixel shift just smooths the visible motion). Each frame: `vy += g`, then `x, y` are advanced by velocity. A fruit despawns when it leaves the bottom of the screen.
- **Cut / miss decision** — when the fruit first crosses the line, the frame checks whether a `swipe_pending` flag is set. If yes → score +1, fruit hidden (CUT). If no → continue falling; when it despawns below the screen, lives −1 (MISS).
- **Frame output** — once per frame the updated fruit params and score are pushed to the FPGA via `FRUIT_WRITE` + `SCORE_WRITE` ioctls; when lives reach 0, `GAME_ST_WRITE` marks `GAME_STATE = 1` and the program exits.

All tuning knobs (threshold, gravity, initial velocity range, launch period) are compile-time constants for now, with the threshold planned as a CLI argument so it can be re-calibrated without rebuilding.

## 4 Resource Budgets

This is a deliberately **low-resource** project: no BRAM, no DSP-intensive math, no off-chip DDR accesses from the FPGA. Budget tables below use the Cyclone V SE 5CSEMA5F31C6 (the DE1-SoC part) as the reference.

### 4.1 FPGA logic — registers (flip-flops)

| Module | Register | Width | Count | Note |
|---|---|---|---|---|
| ultra_sensor_csr | period_cnt | 23 | 23 | 100 ms wrap |
| ultra_sensor_csr | echo_ff1/2/prev | 1 each | 3 | 2-FF sync + edge |
| ultra_sensor_csr | echo_cnt | 32 | 32 | live echo width |
| ultra_sensor_csr | last_echo_cnt | 32 | 32 | latched result |
| ultra_sensor_csr | sample_id | 32 | 32 | monotonic counter |
| ultra_sensor_csr | new_sample | 1 | 1 | dirty bit |
| top_module | echo_s1/s2 | 1 each | 2 | second-stage sync (for VGA indicator) |
| top_module | fruit_x/y/radius/visible | 10/10/10/1 | 31 | game regs |
| top_module | score_reg/game_state_reg | 32/32 | 64 | game regs |
| top_module | readdata | 32 | 32 | Avalon readback latch |
| top_module | VGA_R/G/B | 8/8/8 | 24 | output stage |
| vga_counter | hcount/vcount | 11/10 | 21 | VGA timing |
| **Total** | | | $\approx$ **297 FF** | well under the $\sim$82 000 ALMs on the 5CSEMA5 |

No multi-clock-domain CDC, no FIFO, no metastability hardening beyond the 2-FF synchroniser on `echo_in`.

### 4.2 FPGA memory — BRAM / M10K

**0 bits.** No frame buffer, no sprite table, no look-up ROM. (Contrast: a 640$\times$480$\times$24-bit frame buffer would consume $\sim$7.4 Mbit, more than the $\sim$4.1 Mbit of on-chip M10K the part has — impossible to fit. We avoid the problem entirely by computing pixels combinationally.)

### 4.3 FPGA DSP — 9$\times$9 multipliers

- 2 multipliers for $dx^2 + dy^2$ (10-bit $\times$ 10-bit each) in the fruit hit-test. Synthesis will likely map both into a single DSP block.
- 1 multiplier for the score bar width (`score[5:0] * 10` — will be implemented as `(score[5:0] << 3) + (score[5:0] << 1)` and use no DSP).

Budget: $\leq$ **2 DSP blocks** out of 87 available.

### 4.4 Bus bandwidth

| Channel | Per-frame traffic | @ 30 fps | Headroom |
|---|---|---|---|
| HPS $\rightarrow$ FPGA writes | 4 $\times$ FRUIT_ + 1 $\times$ SCORE = 5 $\times$ 32 b = 20 B | 600 B/s | bridge supports $\sim$100 MB/s, 5 orders of magnitude slack |
| FPGA $\rightarrow$ HPS reads | 3 $\times$ 32 b (STATUS, LAST_ECHO, SAMPLE_ID) + 1 $\times$ 32 b W1C = 16 B | 480 B/s | same |
| Pixel stream | 640$\times$480$\times$60 = 18.4 Mpx/s to the DAC | — | 25 MHz pixel clock, drives DAC directly |

### 4.5 Timing

- 50 MHz single clock. Critical path identified from the P&R timing report as the 22-bit hit-test comparator ($dx^2 + dy^2$ vs $r^2$); at 50 MHz the slack is typically $\geq$ 8 ns on this part. **No pipelining needed.**

## 5 The Hardware/Software Interface

### 5.1 Address window

| Property | Value |
|---|---|
| Physical base address | `0xFF200000` (Lightweight HPS-to-FPGA bridge) |
| Window span | `0x40` (64 bytes, 10 $\times$ 32-bit words) |
| Address units (Qsys) | WORDS — external `address[3:0]` is a word index |
| Read wait time | 1 cycle (Avalon `readWaitTime = 1`) |
| Write wait time | 0 cycles |
| `waitrequest` | tied to 0 — every transaction completes deterministically |

The driver maps this window once with `of_iomap()` and stores the resulting kernel virtual address in `virtbase`. All register accesses go through `ioread32(virtbase + off)` / `iowrite32(val, virtbase + off)`.

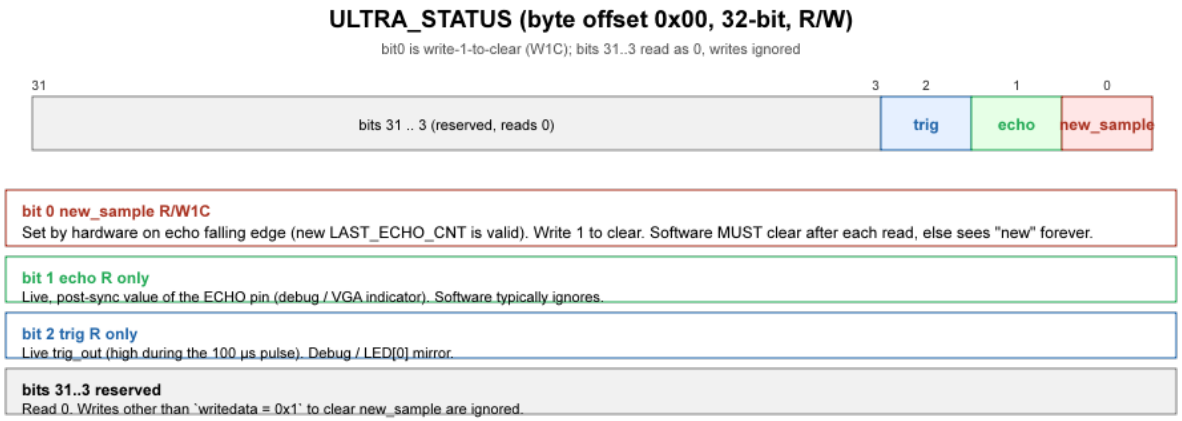


Figure 5: ULTRA_STATUS bit-field layout.

## 5.2 Register map

| Byte Offset | Word | Name | Access | Width | Reset | Purpose |
|---|---|---|---|---|---|---|
| 0x00 | 0 | ULTRA_STATUS | R / W1C | 3 of 32 | 0 | bit0 = new_sample, bit1 = echo, bit2 = trig; write 0x1 clears new_sample |
| 0x04 | 1 | LAST_ECHO_CNT | R | 32 | 0 | Latched echo-high duration, 50 MHz cycles; distance(cm) = val / 2900 |
| 0x08 | 2 | CURR_ECHO_CNT | R | 32 | 0 | Live counter — debug only, not read by game |
| 0x0C | 3 | SAMPLE_ID | R | 32 | 0 | Incremented once per completed measurement; software's primary freshness signal |
| 0x10 | 4 | FRUIT_X | W | 10 of 32 | 320 | Fruit centre x, 0..639 |
| 0x14 | 5 | FRUIT_Y | W | 10 of 32 | 240 | Fruit centre y, 0..479 |
| 0x18 | 6 | FRUIT_RADIUS | W | 10 of 32 | 30 | Fruit radius, 20..35 |
| 0x1C | 7 | FRUIT_CTRL | W | 1 of 32 | 0 | bit0 = visible (rendered only when 1) |
| 0x20 | 8 | SCORE | W | 32 | 0 | Score; only low 6 bits drive the gold bar (10 px/point, $\leq$ 63 points) |
| 0x24 | 9 | GAME_STATE | W | 32 | 0 | 0 = running, 1 = game over (recorded; not yet rendered) |

## 5.3 Bit-level detail of the one register with multiple fields

- **bit 0 `new_sample` (R / W1C)** — Set by hardware on the echo falling edge. Indicates "LAST_ECHO_CNT holds a measurement the software has not yet acknowledged." Software **must** write `0x1` back to this register after a successful read, otherwise every poll will keep seeing `new_sample = 1` forever. Writing any other value to this register is ignored.
- **bit 1 echo (R)** — live synchronised `echo_in`. Used by the VGA indicator and the `LED[1]` mirror. Software ignores this bit.
- **bit 2 `trig` (R)** — live `trig_out` (high during the 100 $\mu$s pulse). Used by `LED[0]`. Software ignores this bit.
- **bits 31:3** — reserved, read as 0.

The other six writable registers are *single-field*: the entire 32-bit write is passed through to the destination (masked to its physical width inside the module). A write to a width-10 register with `writedata > 0x3FF` silently truncates.

### `ultra_sensor_csr` local Avalon-MM interface and bit interpretation

`ultra_sensor_csr` is a simple Avalon-MM slave. Its local interface consists of `s0_address`, `s0_read`, `s0_write`, `s0_chipselect`, `s0_writedata`, `s0_readdata`, and `s0_waitrequest`, together with `clk` and active-low `reset_n`. The module exposes four readable registers: ULTRA_STATUS, LAST_ECHO_CNT, CURR_ECHO_CNT, and SAMPLE_ID. The only supported write operation is a write-one-to-clear action on ULTRA_STATUS[0], which clears new_sample; all other writes are ignored. Since `s0_waitrequest` is tied low, the slave never stalls the bus.

| Signal | Width | Direction | Meaning |
|---|---|---|---|
| clk | 1 | in | 50 MHz module clock. |
| reset_n | 1 | in | Active-low reset for the entire CSR block. |
| s0_address | 2 | in | Local word address inside ultra_sensor_csr: 00→ULTRA_STATUS, 01→LAST_ECHO_CNT, 10→CURR_ECHO_CNT, 11→SAMPLE_ID. |
| s0_read | 1 | in | Read strobe from the Avalon-MM master. |
| s0_write | 1 | in | Write strobe from the Avalon-MM master. |
| s0_chipselect | 1 | in | Indicates that this slave is selected for the current transaction. |
| s0_writedata | 32 | in | Write data bus. Only s0_writedata[0] is currently used. |
| s0_readdata | 32 | out | Read data bus returned to the master. |
| s0_waitrequest | 1 | out | Always 0; this slave never stalls the bus. |
| echo_in | 1 | in | Raw asynchronous echo signal from the HC-SR04 sensor. |
| trig_out | 1 | out | Trigger pulse driven to the HC-SR04 sensor. |

### Local Avalon-MM signals

**Register bit interpretation** Only ULTRA_STATUS is a true bit-field register. The other exported registers are numeric counters rather than collections of independent control/status bits.

**ULTRA_STATUS bit-field**

| Bits | Name | Access | Meaning | Hardware behavior | Software behavior |
|---|---|---|---|---|---|
| [0] | new_sample | R/W1C | Indicates that LAST_ECHO_CNT holds a newly completed measurement not yet acknowledged by software. | Set to 1 on echo_fall, when the current echo width is latched into LAST_ECHO_CNT. | Software writes 1 to clear it after consuming the sample. |
| [1] | echo | R | Live synchronized echo level. | Reflects the current value of the synchronized echo signal. | Normally ignored by the game logic; useful for debug / LED / VGA indicator. |
| [2] | trig | R | Live trigger output level. | High during the trigger pulse interval. | Normally ignored by the game logic; useful for debug / LED. |
| [31:3] | Reserved | R | Unused bits. | Always read as 0. | Must be ignored by software. |

**Numeric counter registers**

| Register | Width | Bit meaning | Note |
|---|---|---|---|
| LAST_ECHO_CNT | 32 | bit[n] contributes binary weight $2^n$ to the full latched count value. | Latched echo-high duration in 50 MHz clock cycles. This is a numeric register, not a bit-field register. |
| CURR_ECHO_CNT | 32 | bit[n] contributes binary weight $2^n$ to the full live count value. | Live counter value while echo is high; used mainly for debugging. |
| SAMPLE_ID | 32 | bit[n] contributes binary weight $2^n$ to the full sample sequence number. | Incremented by 1 once per completed measurement; used by software as the primary freshness signal. |

## 5.4 Software API (user space)

```c
/* top_module.h */
#include <linux/ioctl.h>

typedef struct {
    unsigned int new_sample;      /* 0 or 1 */
    unsigned int echo;
    unsigned int trig;
    unsigned int last_echo_cnt;   /* 50 MHz cycles */
    unsigned int sample_id;
} ultra_status_t;

typedef struct {
    unsigned int x;               /* 0..639 */
    unsigned int y;               /* 0..479 */
    unsigned int radius;          /* 20..35 */
    unsigned int visible;         /* 0 or 1 */
} fruit_params_t;

#define ULTRA_READ     _IOR('F', 1, ultra_status_t)
#define FRUIT_WRITE    _IOW('F', 2, fruit_params_t)
#define SCORE_WRITE    _IOW('F', 3, unsigned int)
#define GAME_ST_WRITE  _IOW('F', 4, unsigned int)
```

**Behavioural contract of each ioctl:**

| ioctl | Kernel side effect on FPGA |
|---|---|
| ULTRA_READ | ioread32 STATUS, LAST_ECHO_CNT, SAMPLE_ID → copy_to_user; then iowrite32(1, STATUS) to clear new_sample |
| FRUIT_WRITE | copy_from_user; four iowrite32s to FRUIT_X/Y/RADIUS/CTRL |
| SCORE_WRITE | copy_from_user; one iowrite32 to SCORE |
| GAME_ST_WRITE | copy_from_user; one iowrite32 to GAME_STATE |

## 5.5 Hardware module interface

```
module top_module (
    input  logic        clk,                // 50 MHz
    input  logic        reset,              // active-high

    // Avalon-MM slave (addressUnits = WORDS)
    input  logic        chipselect,
```

```
    input  logic [3:0]  address,
    input  logic        read,
    output logic [31:0] readdata,
    input  logic        write,
    input  logic [31:0] writedata,

    // VGA to ADV7123
    output logic [7:0]  VGA_R, VGA_G, VGA_B,
    output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N,

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

    // HC-SR04 (conduits)
    input  logic        echo_in,            // GPIO_0[0], post-divider
    output logic        trig_out            // GPIO_1[0]
);
```

Instance hierarchy (matches the `soc_system.qsys`):

```
de1_soc_top.v         (Quartus top-level wrapper)
 |-- PLL_0002         (from GHRD; 50->66 MHz for the HPS, unchanged)
 \-- soc_system       (Qsys-generated)
      |-- hps_0       (Cortex-A9 + SDRAM ctl + peripherals -- GHRD default)
      |-- clk_0       (50 MHz clock source)
      \-- top_module  (this project's custom peripheral)
           |-- ultra_sensor_csr (ultra_i)
           \-- vga_counter      (counter_0)
```

## 5.6 Pinout

| Signal | FPGA pin | Voltage | Notes |
|---|---|---|---|
| `echo_in` | GPIO_0[0] | 3.3 V | 5 V sensor output → 1 kΩ + 2 kΩ divider → ∼3.3 V |
| `trig_out` | GPIO_1[0] | 3.3 V | Directly drives HC-SR04 TRIG (threshold ∼0.8 × Vcc ≈ 2.4 V) |
| VGA_R/G/B[7:0], HS/VS/BLANK_N/SYNC_N, VGA_CLK | ADV7123 (on-board) | — | — |
| `LED[0]` | LEDR0 | — | Mirrors `trig_out` (blinks ∼10 Hz) |
| `LED[1]` | LEDR1 | — | Mirrors `echo_in` |

## 5.7 Future improvements

- **Add second sensor** — In order to detect hand wave more precise, we will try to add another sensor. A successful cutting action will be detected when these two sensor both report distance decrese and there's a time delay between them.
- **10 Hz sample rate** — 30 fps game loop sees a new sample every 3 frames (∼100 ms). This is perceptible. Fall-back: change the trigger period from 5 000 000 to 1 000 000 cycles (20 ms) to get 50 Hz; the FPGA cost is 0 because `period_cnt` is already 23 bits.