CSEE4840 Presentation: PlantHealth

Linus Lei, Daolin Li , Gurleen Kalra

Introduction

- Modern agriculture requires real-time, low-cost plant monitoring for better crop management.
- Traditional manual methods are slow, subjective, and not suitable for continuous monitoring.
- This project presents an FPGA–HPS based embedded system on the DE1-SoC for plant image acquisition and analysis using an OV7670 camera.
- FPGA handles low-level real-time tasks like image capture, conversion, buffering, VGA display, and ExG computation.

- HPS performs high-level processing such as k-means clustering, segmentation, and plant health classification.
- The system provides real-time VGA output showing original and processed images with health status.
- The design demonstrates an efficient hardware–software co-design approach for smart agriculture.

Background

- Smart agriculture is increasingly adopting embedded vision systems for automated crop and plant monitoring.
- Traditional plant-health assessment methods rely on manual visual inspection, which is inconsistent and not scalable for large farms.
- Recent developments in FPGA-based embedded platforms, low-cost CMOS image sensors, and computer vision techniques have enabled real-time agricultural monitoring.
- Image-based vegetation analysis methods such as Excess Green (ExG) and clustering techniques help in identifying plant regions and health status.
- Co-processing architectures combining FPGA (for real-time processing) and processor systems (for complex algorithms) are widely used for efficient embedded vision systems.
- These advancements motivate the development of low-cost, real-time, and interpretable plant-health monitoring systems for precision agriculture.

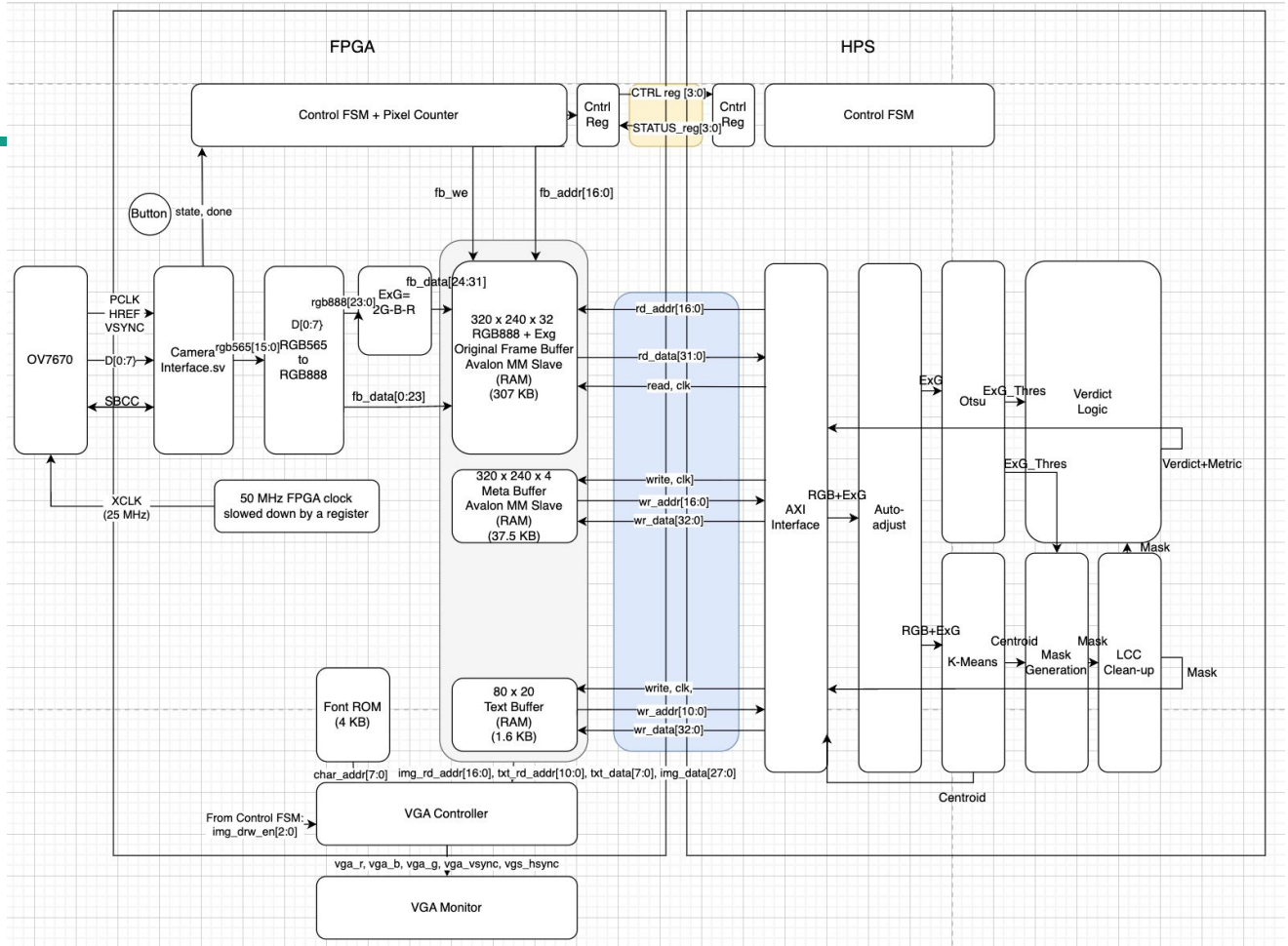
Literature Review

Previous Work

- **Plant Disease Detection:** Sankaran et al. reviewed computer-vision techniques for automated plant disease and stress detection in precision agriculture.
- **Vegetation Indices:** Woebbecke et al. demonstrated the effectiveness of Excess Green (ExG) for separating vegetation from soil and background.
- **FPGA Image Processing:** Donald Bailey presented FPGA architectures for real-time embedded image-processing systems.
- **OV7670 Camera Systems:** Multiple FPGA-based embedded vision projects have used the OV7670 Camera Module for real-time image capture and VGA display applications.
- **Hardware–Software Co-Design:** Cyclone V SoC FPGA platforms have been widely used for heterogeneous embedded vision systems combining FPGA acceleration with processor-based computation.
- **K-Means Segmentation:** MacQueen introduced the k-means clustering algorithm, which later became widely used for image segmentation and agricultural imaging applications.

This project builds upon these earlier works by integrating FPGA-based image acquisition, hardware ExG computation, clustering-based plant segmentation, and VGA visualization into a unified real-time plant-monitoring system.

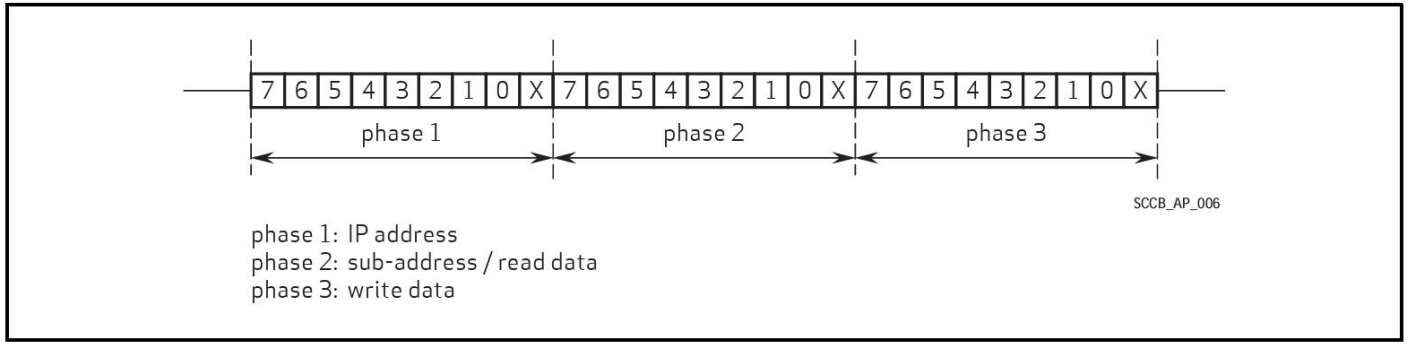
Block Diagram



Interfacing the Camera - SCCB

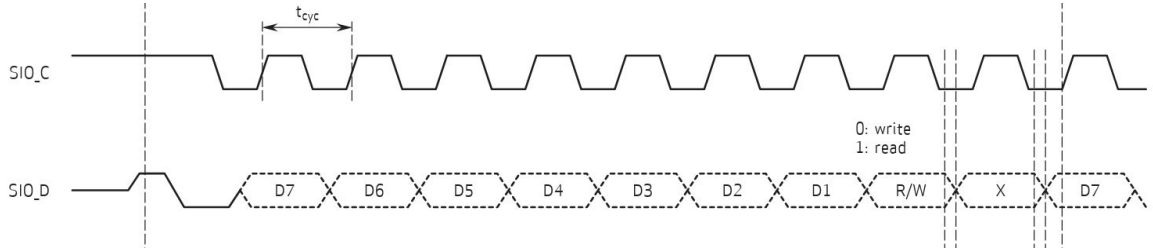
- Two wire protocol: SIOC, SIOD
- Three Phase Write:

Figure 3-4 Transmission Phases




```
assign   siod = (step == 13 || step == 14 || step == 24 || step == 25 || step == 35 || step == 36 ||
                 step == 52 || step == 53 || step >= 54 && step <= 62) ? 1'bz : bit_out;
```

Interfacing the Camera - SCCB



-Bit Bang Style controller

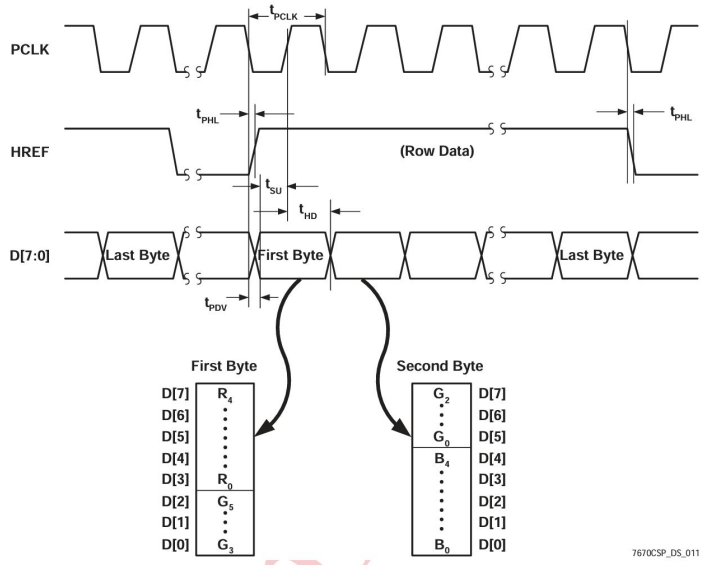
```
// Write data. This concludes 3-phase write transaction.
7'd26:
  bit_out <= data_in[7];
7'd27:
  bit_out <= data_in[6];
7'd28:
  bit_out <= data_in[5];
7'd29:
  bit_out <= data_in[4];
7'd30:
  bit_out <= data_in[3];
7'd31:
  bit_out <= data_in[2];
7'd32:
  bit_out <= data_in[1];
7'd33:
  bit_out <= data_in[0];
7'd34:
  bit_out <= 0;
7'd35:
  ack_err3 <= siod;
7'd36:
  bit_out <= 0;
```

Interfacing the Camera - SCCB

- Set the output resolution to QVGA (320x240)
- Set the output format to RGB565
- Enable AWB
- Adjust AWB

Interfacing the Camera - Receiving

Figure 11 RGB 565 Output Timing Diagram




```
if (frameValid == 1 && vsync_i == 0 && href_i == 1)
begin
  if (odd == 0)
  begin
    pixel_o[15:8] <= d;      // 1st byte = high (RRRRR
  end
  else
  begin
    pixel_o[7:0] <= d;       // 2nd byte = low  (GGGBB
    pixelReady_o <= 1;
  end
  odd <= ~odd;
end
else
begin
```

Preprocessing-FPGA

-Bit Replication to RGB888

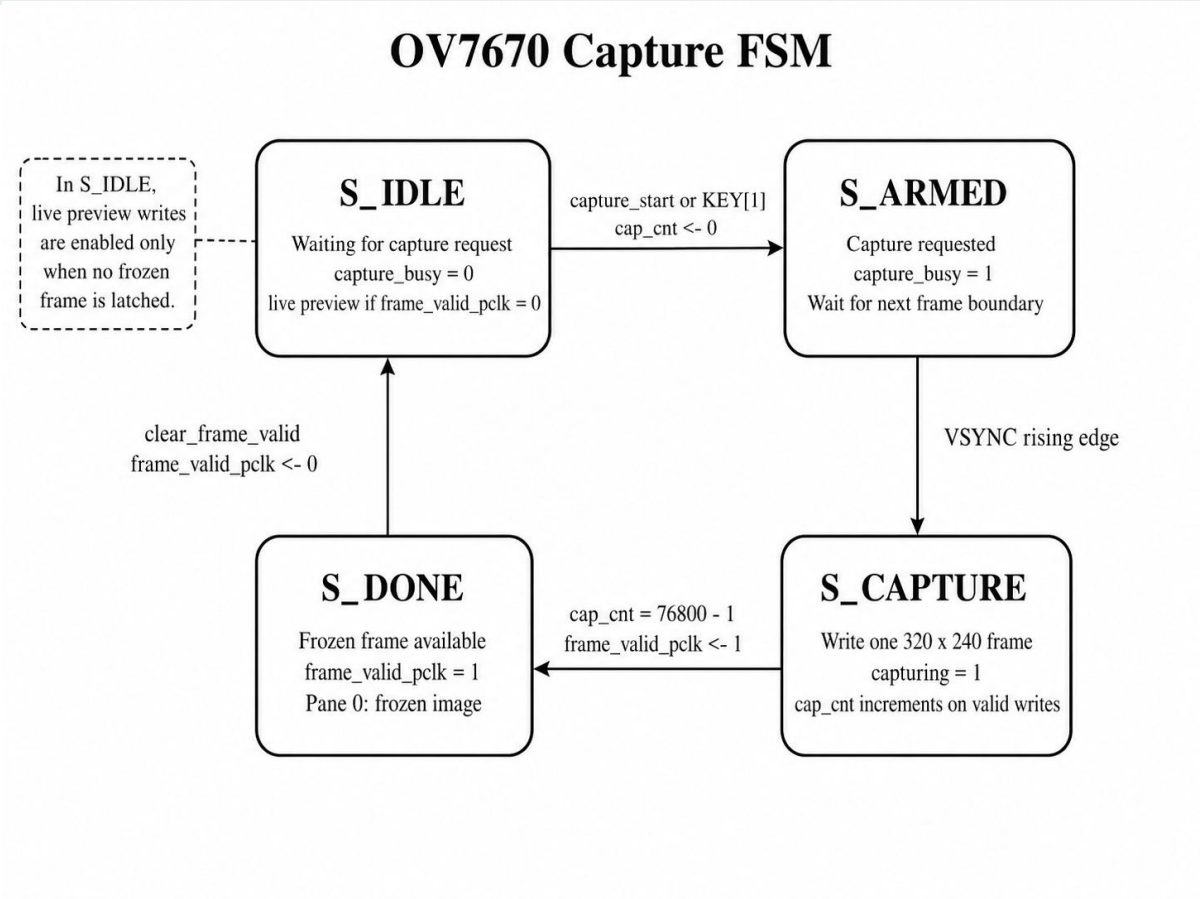
-Calculate ExG (8bit):

ExG = 2G-R-B

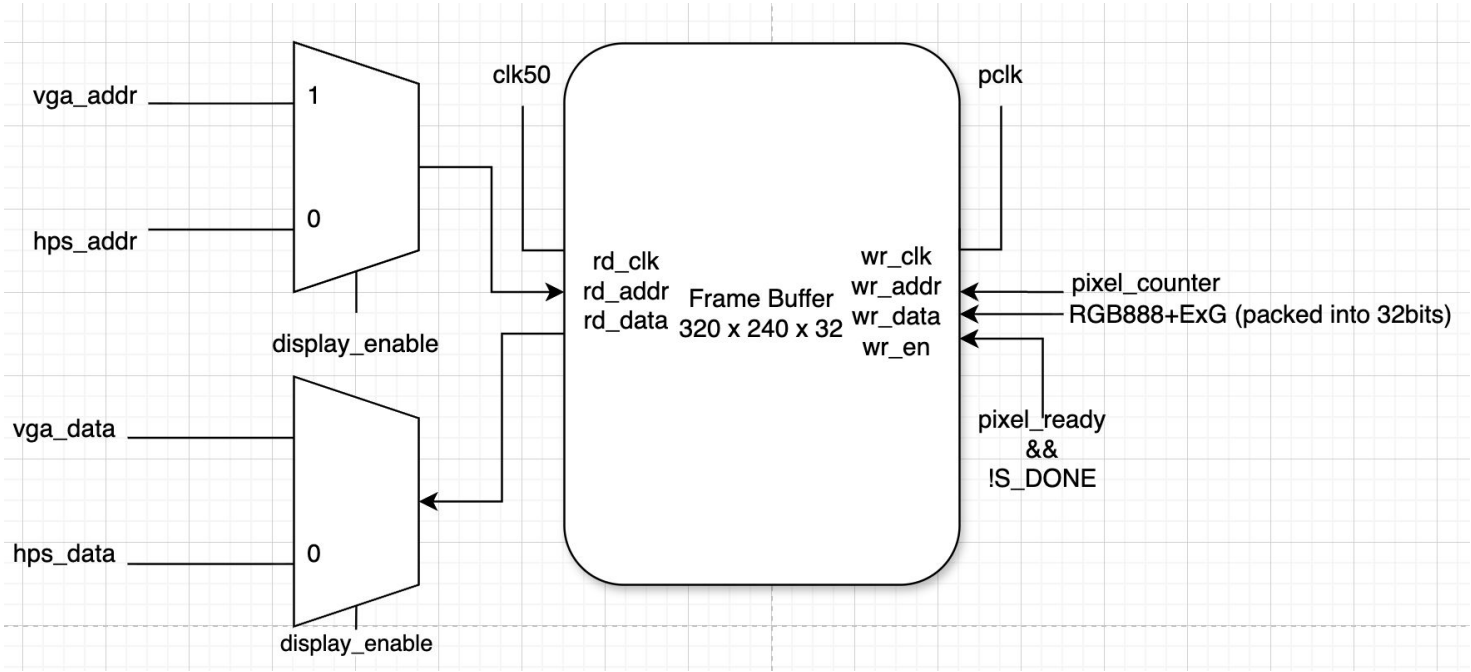
-Store to Buffer

```
wire [4:0] raw_r5 = pixel_data[15:11];
wire [5:0] raw_g6 = pixel_data[10:5];
wire [4:0] raw_b5 = pixel_data[4:0];

wire [7:0] r8 = {raw_r5, raw_r5[4:2]};
wire [7:0] g8 = {raw_g6, raw_g6[5:4]};
wire [7:0] b8 = {raw_b5, raw_b5[4:2]};

wire signed [9:0] exg_full = $signed({2'b0, g8}) * 2
                           - $signed({2'b0, r8})
                           - $signed({2'b0, b8});
wire [7:0] exg8 = (exg_full < 0)   ? 8'd0 :
                  (exg_full > 255) ? 8'd255 :
                  exg_full[7:0];

wire [31:0] orig_wr_data = {exg8, r8, g8, b8};
```

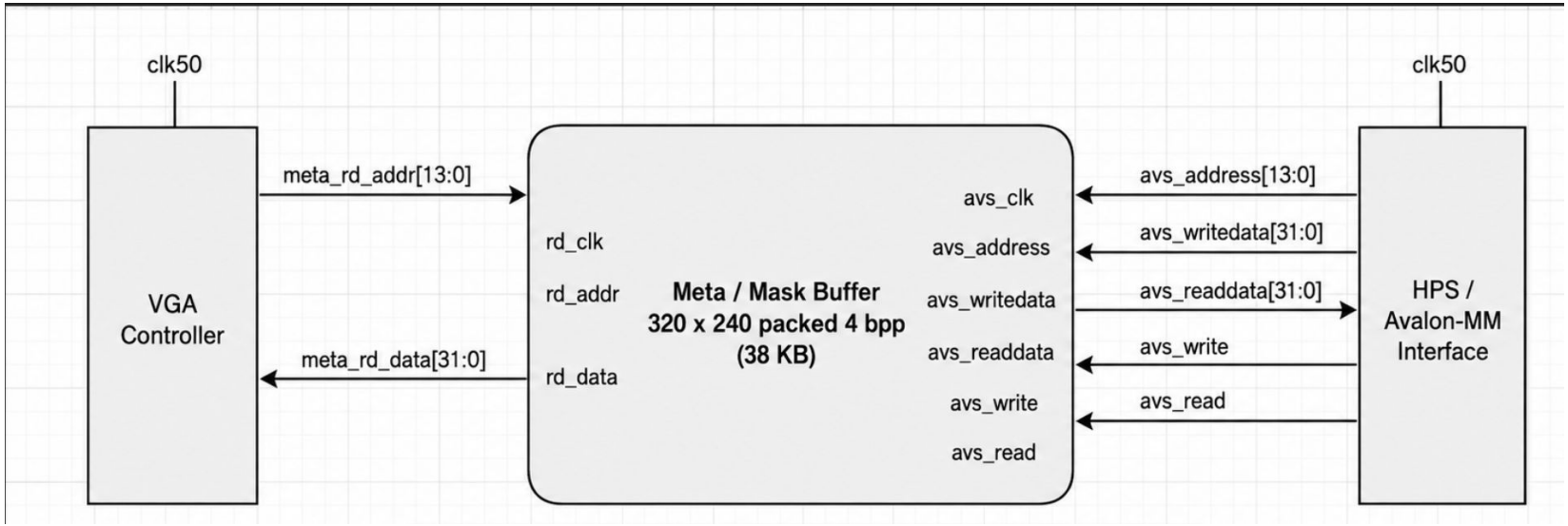
FSM


OV7670 Capture FSM
In S_IDLE, live preview writes are enabled only when no frozen frame is latched.
S_IDLE
Waiting for capture request
capture_busy = 0
live preview if frame_valid_pclk = 0
capture_start or KEY[1]
cap_cnt <- 0
S_ARMED
Capture requested
capture_busy = 1
Wait for next frame boundary
VSYNC rising edge
S_CAPTURE
Write one 320 x 240 frame
capturing = 1
cap_cnt increments on valid writes
cap_cnt = 76800 - 1
frame_valid_pclk <- 1
S_DONE
Frozen frame available
frame_valid_pclk = 1
Pane 0: frozen image
clear_frame_valid
frame_valid_pclk <- 0

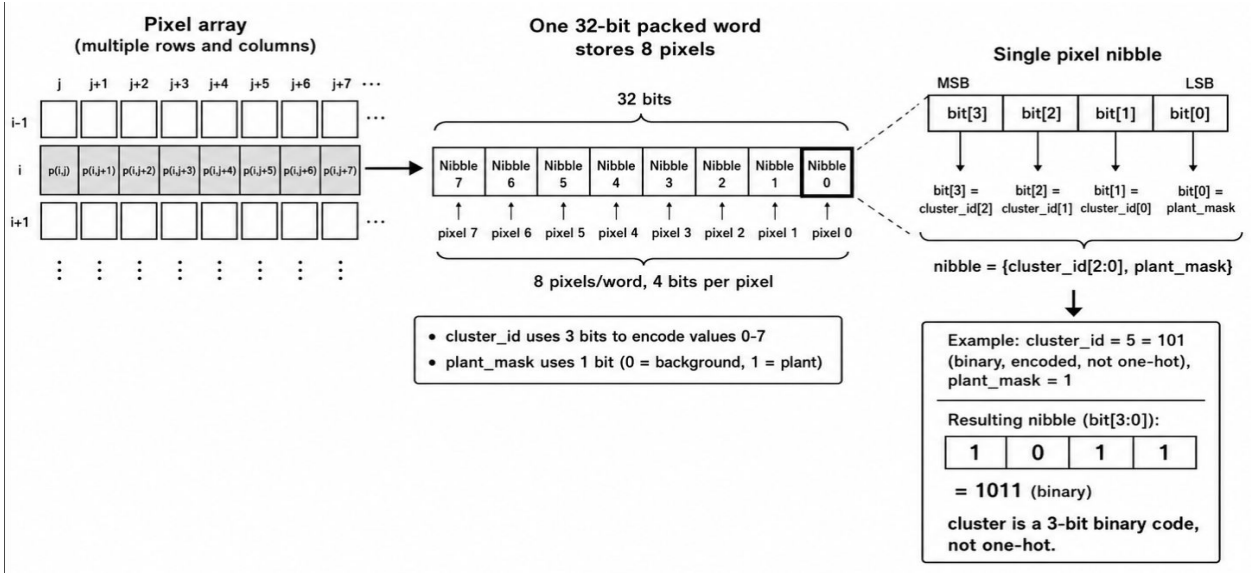
Frame Buffers - M10K Memory Blocks



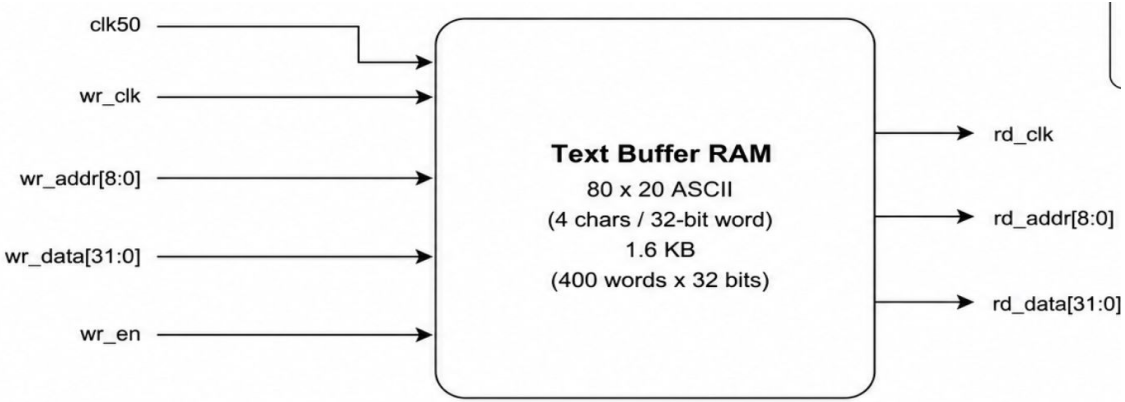
Meta Buffer


clk50
clk50
VGA
Controller
meta_rd_addr[13:0]
meta_rd_data[31:0]
rd_clk
rd_addr
rd_data
Meta / Mask Buffer
320 x 240 packed 4 bpp
(38 KB)
avs_clk
avs_address
avs_writedata
avs_readdata
avs_write
avs_read
avs_address[13:0]
avs_writedata[31:0]
avs_readdata[31:0]
avs_write
avs_read
HPS /
Avalon-MM
Interface

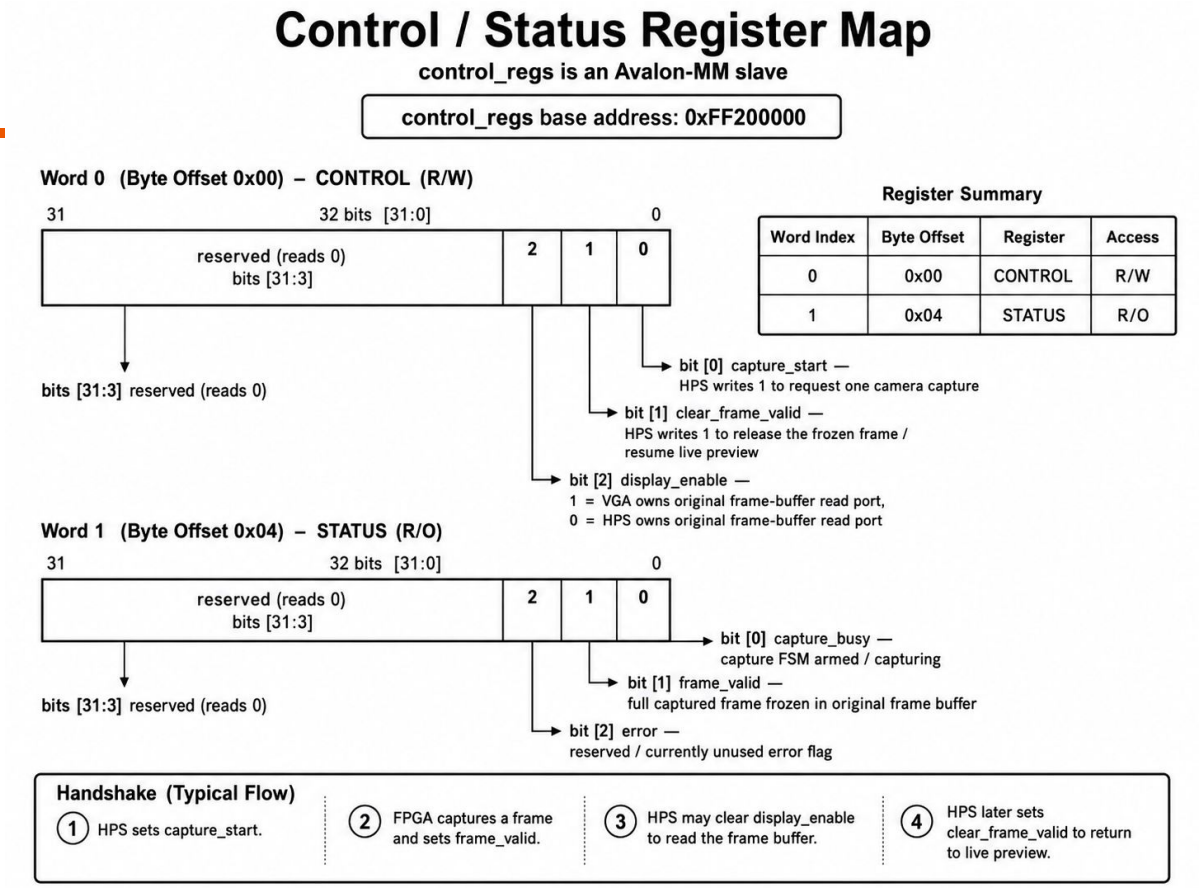
Meta Buffer


Pixel array
(multiple rows and columns)
j
j+1
j+2
j+3
j+4
j+5
j+6
j+7
i-1
i
i+1
p(i,j)
p(i,j+1)
p(i,j+2)
p(i,j+3)
p(i,j+4)
p(i,j+5)
p(i,j+6)
p(i,j+7)
One 32-bit packed word
stores 8 pixels
32 bits
Nibble 7
Nibble 6
Nibble 5
Nibble 4
Nibble 3
Nibble 2
Nibble 1
Nibble 0
pixel 7
pixel 6
pixel 5
pixel 4
pixel 3
pixel 2
pixel 1
pixel 0
8 pixels/word, 4 bits per pixel
cluster_id uses 3 bits to encode values 0-7
plant_mask uses 1 bit (0 = background, 1 = plant)
Single pixel nibble
MSB
LSB
bit[3]
bit[2]
bit[1]
bit[0]
bit[3] = cluster_id[2]
bit[2] = cluster_id[1]
bit[1] = cluster_id[0]
bit[0] = plant_mask
nibble = {cluster_id[2:0], plant_mask}
Example: cluster_id = 5 = 101 (binary, encoded, not one-hot), plant_mask = 1
Resulting nibble (bit[3:0]):
1
0
1
1
= 1011 (binary)
cluster is a 3-bit binary code, not one-hot.

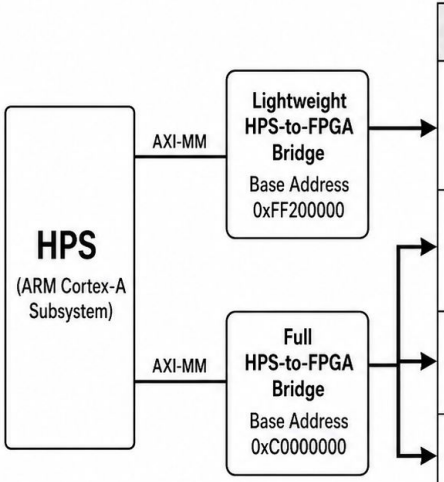
Text Buffer


clk50
wr_clk
wr_addr[8:0]
wr_data[31:0]
wr_en
Text Buffer RAM
80 x 20 ASCII
(4 chars / 32-bit word)
1.6 KB
(400 words x 32 bits)
rd_clk
rd_addr[8:0]
rd_data[31:0]
32-bit Word Format (4 ASCII Characters)
31
24
23
16
15
8
7
0
Char 3
(bits 31:24)
Char 2
(bits 23:16)
Char 1
(bits 15:8)
Char 0
(bits 7:0)
MSB
LSB

Hardware Software Interface



Hardware-Software Memory Map

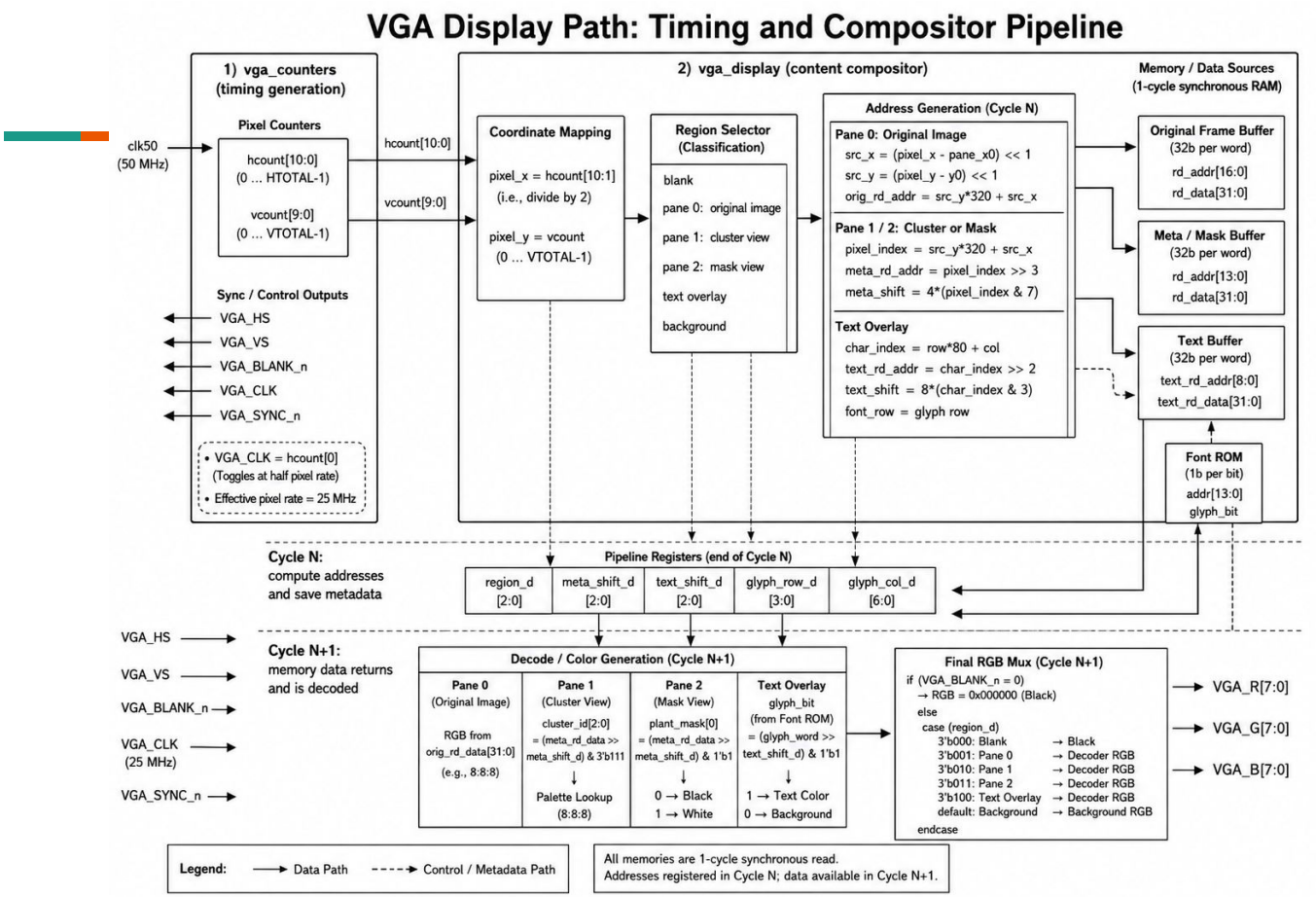


Base Address	Module	Access	Size/Span	Purpose
0xFF200000	control_regs	R/W + R/O	2 × 32-bit words	Capture / display control and FPGA status
0xC0000000	frame_buffer_avmm	Read-only	0x4B000 bytes	Original captured frame buffer
0xC0100000	meta_buffer_avmm	Read / write	0x9600 bytes	Packed cluster ID + plant mask
0xC0200000	text_buffer_avmm	Read / write	0x640 bytes	Packed ASCII text overlay

Note:
All custom slaves are 32-bit word-addressed Avalon-MM slaves. In software, volatile uint32_t* index n corresponds to byte offset 4n.

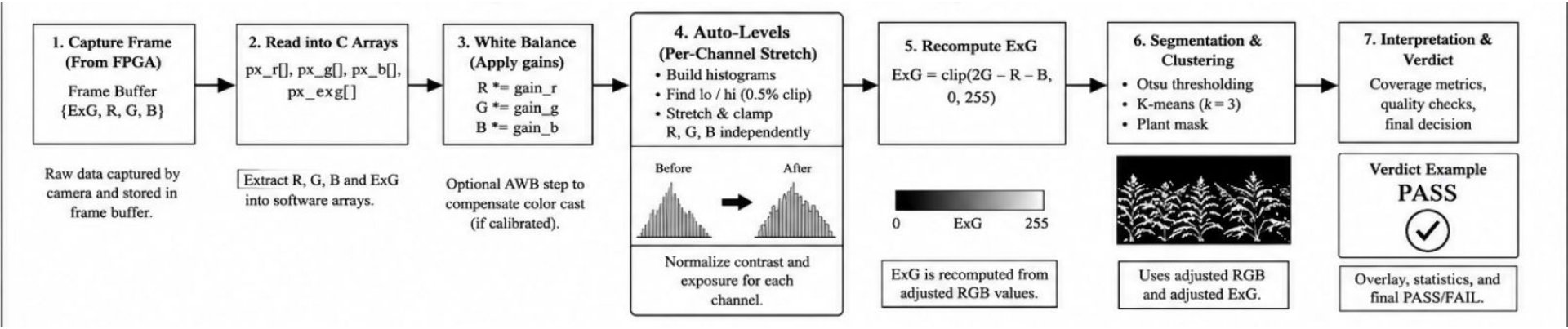
Legend (Access Types):
R/O = read only
R/W = read/write

VGA Display


VGA Display Path: Timing and Compositor Pipeline
1) vga_counters (timing generation)
clk50 (50 MHz)
Pixel Counters
hcount[10:0] (0 ... HTOTAL-1)
vcount[9:0] (0 ... VTOTAL-1)
hcount[10:0]
vcount[9:0]
Sync / Control Outputs
VGA_HS
VGA_VS
VGA_BLANK_n
VGA_CLK
VGA_SYNC_n
• VGA_CLK = hcount[0] (Toggles at half pixel rate)
• Effective pixel rate = 25 MHz
2) vga_display (content compositor)
Coordinate Mapping
pixel_x = hcount[10:1] (i.e., divide by 2)
pixel_y = vcount (0 ... VTOTAL-1)
Region Selector (Classification)
blank
pane 0: original image
pane 1: cluster view
pane 2: mask view
text overlay
background
Address Generation (Cycle N)
Pane 0: Original Image
src_x = (pixel_x - pane_x0) << 1
src_y = (pixel_y - y0) << 1
orig_rd_addr = src_y*320 + src_x
Pane 1 / 2: Cluster or Mask
pixel_index = src_y*320 + src_x
meta_rd_addr = pixel_index >> 3
meta_shift = 4*(pixel_index & 7)
Text Overlay
char_index = row*80 + col
text_rd_addr = char_index >> 2
text_shift = 8*(char_index & 3)
font_row = glyph row
Memory / Data Sources (1-cycle synchronous RAM)
Original Frame Buffer (32b per word)
rd_addr[16:0]
rd_data[31:0]
Meta / Mask Buffer (32b per word)
rd_addr[13:0]
rd_data[31:0]
Text Buffer (32b per word)
text_rd_addr[8:0]
text_rd_data[31:0]
Font ROM (1b per bit)
addr[13:0]
glyph_bit
Cycle N: compute addresses and save metadata
Pipeline Registers (end of Cycle N)
region_d [2:0]
meta_shift_d [2:0]
text_shift_d [2:0]
glyph_row_d [3:0]
glyph_col_d [6:0]
VGA_HS
VGA_VS
VGA_BLANK_n
VGA_CLK (25 MHz)
VGA_SYNC_n
Cycle N+1: memory data returns and is decoded
Decode / Color Generation (Cycle N+1)
Pane 0 (Original Image)
RGB from orig_rd_data[31:0] (e.g., 8:8:8)
Pane 1 (Cluster View)
cluster_id[2:0] = (meta_rd_data >> meta_shift_d) & 3'b111
Palette Lookup (8:8:8)
Pane 2 (Mask View)
plant_mask[0] = (meta_rd_data >> meta_shift_d) & 1'b1
0 → Black
1 → White
Text Overlay
glyph_bit (from Font ROM) = (glyph_word >> text_shift_d) & 1'b1
1 → Text Color
0 → Background
Final RGB Mux (Cycle N+1)
if (VGA_BLANK_n = 0)
→ RGB = 0x000000 (Black)
else
case (region_d)
3'b000: Blank → Black
3'b001: Pane 0 → Decoder RGB
3'b010: Pane 1 → Decoder RGB
3'b011: Pane 2 → Decoder RGB
3'b100: Text Overlay → Decoder RGB
default: Background → Background RGB
endcase
VGA_R[7:0]
VGA_G[7:0]
VGA_B[7:0]
Legend: Data Path Control / Metadata Path
All memories are 1-cycle synchronous read.
Addresses registered in Cycle N; data available in Cycle N+1.

Software

Software Pipeline



Auto Level

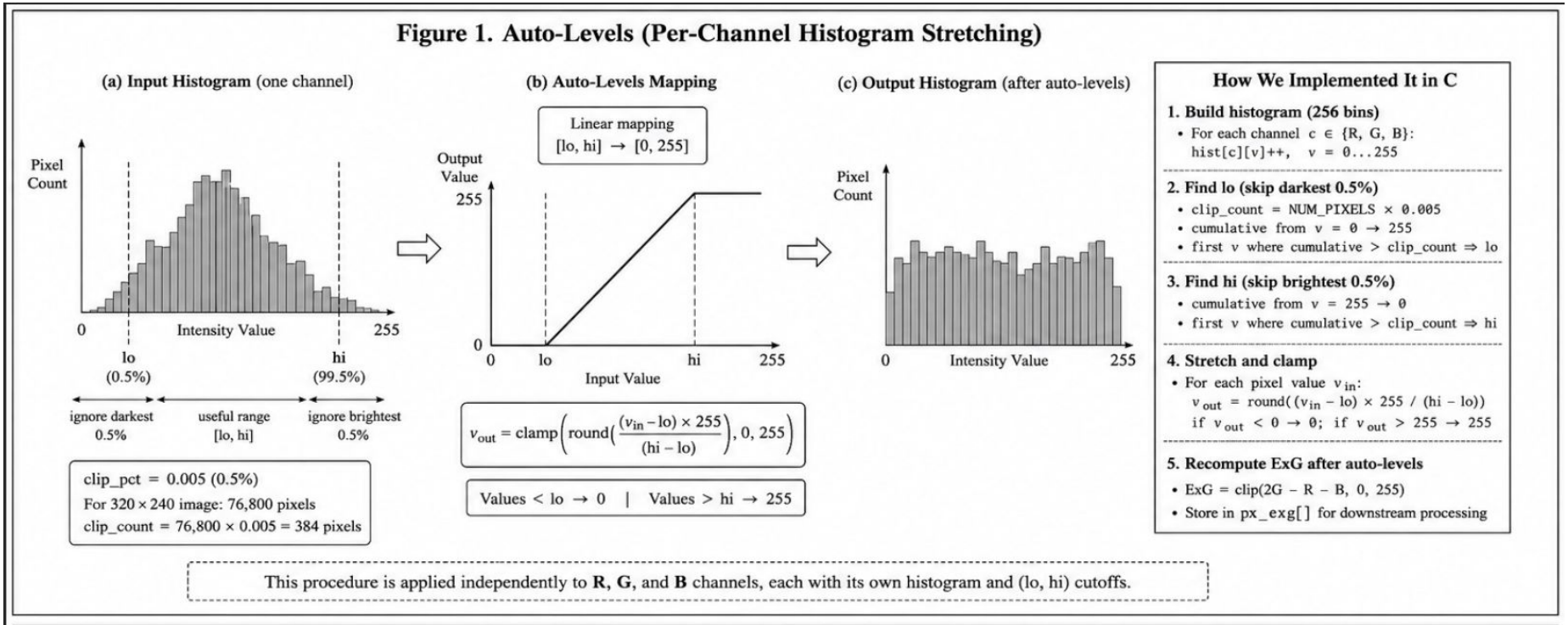
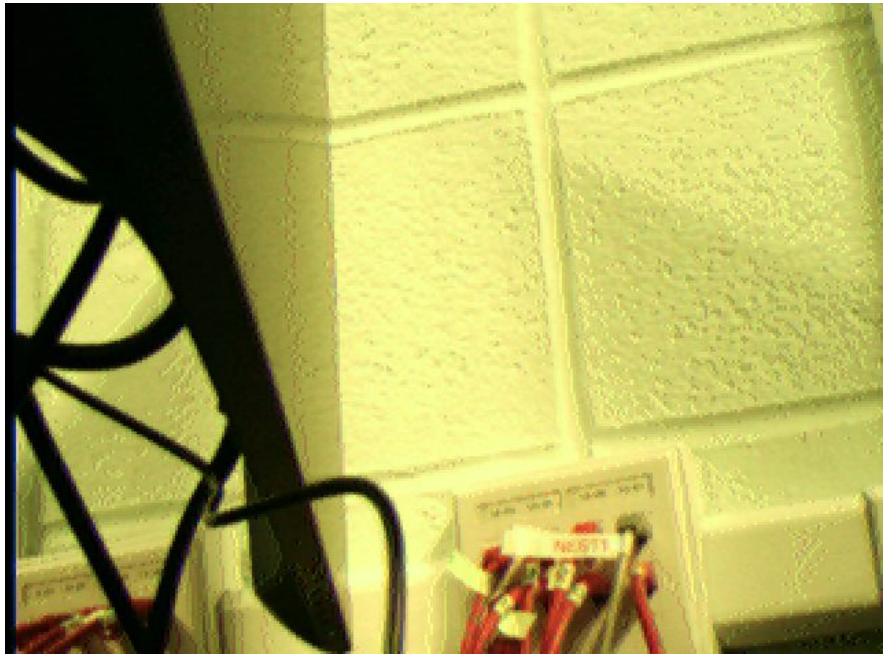
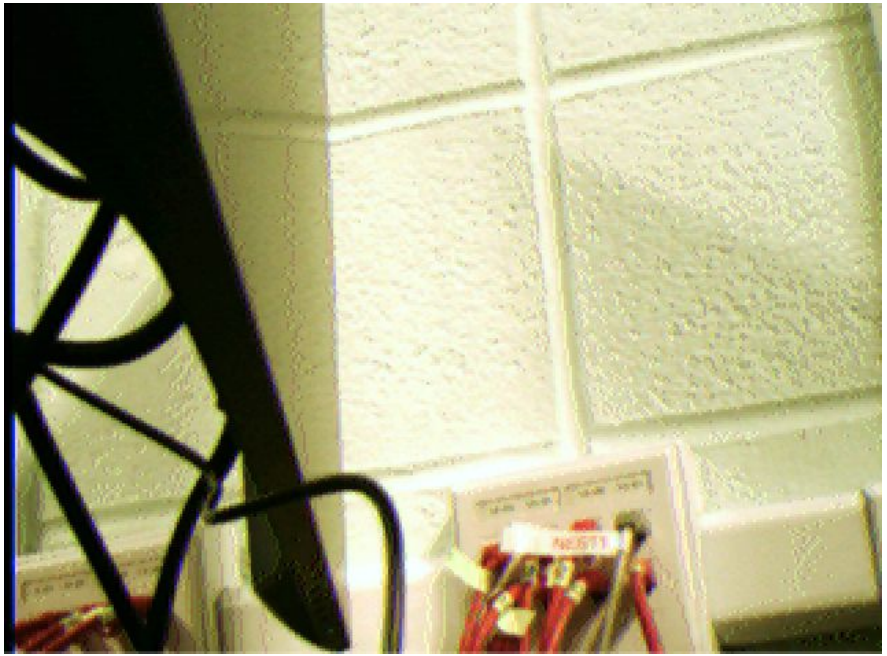


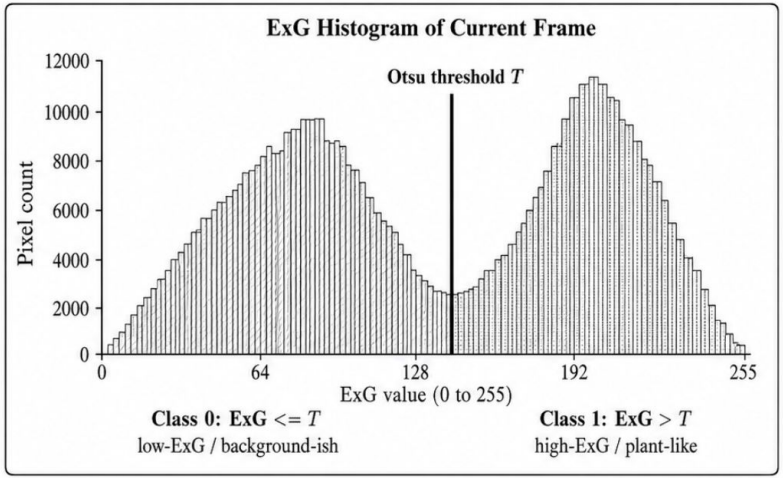
Figure 1. Auto-Levels (Per-Channel Histogram Stretching)
(a) Input Histogram (one channel)
Pixel Count
0
Intensity Value
255
lo
(0.5%)
hi
(99.5%)
ignore darkest 0.5%
useful range [lo, hi]
ignore brightest 0.5%
clip_pct = 0.005 (0.5%)
For 320 × 240 image: 76,800 pixels
clip_count = 76,800 × 0.005 = 384 pixels
(b) Auto-Levels Mapping
Linear mapping
[lo, hi] → [0, 255]
Output Value
255
0
lo
hi
Input Value
$v_{out} = \text{clamp}\left(\text{round}\left(\frac{(v_{in} - lo) \times 255}{(hi - lo)}\right), 0, 255\right)$
Values < lo → 0 | Values > hi → 255
(c) Output Histogram (after auto-levels)
Pixel Count
0
Intensity Value
255
How We Implemented It in C
1. Build histogram (256 bins)
• For each channel c ∈ {R, G, B}: hist[c][v]++, v = 0...255
2. Find lo (skip darkest 0.5%)
• clip_count = NUM_PIXELS × 0.005
• cumulative from v = 0 → 255
• first v where cumulative > clip_count ⇒ lo
3. Find hi (skip brightest 0.5%)
• cumulative from v = 255 → 0
• first v where cumulative > clip_count ⇒ hi
4. Stretch and clamp
• For each pixel value v_{in}: v_{out} = round((v_in − lo) × 255 / (hi − lo)) if v_{out} < 0 → 0; if v_{out} > 255 → 255
5. Recompute ExG after auto-levels
• ExG = clip(2G − R − B, 0, 255)
• Store in px_exg[] for downstream processing
This procedure is applied independently to R, G, and B channels, each with its own histogram and (lo, hi) cutoffs.

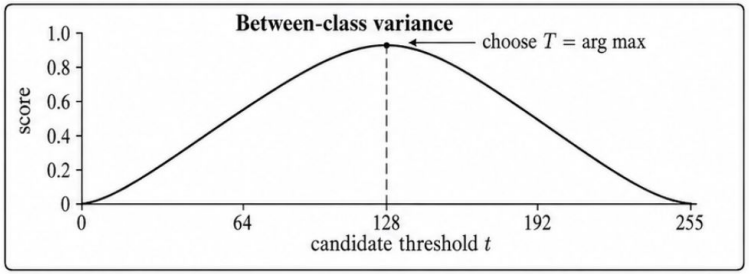
Auto Level





Otsu's Method





Otsu's Criterion

$$\sigma_b^2(t) = w_b w_f (\mu_b - \mu_f)^2$$

Select the threshold that maximizes separation between the two classes.

Otsu Workflow

1. Build ExG histogram hist[0..255]
2. Test thresholds $t = 0..255$
3. Pick T with maximum between-class variance

Use of Otsu Threshold

In this project, T is used as an adaptive ExG cutoff for cluster interpretation.

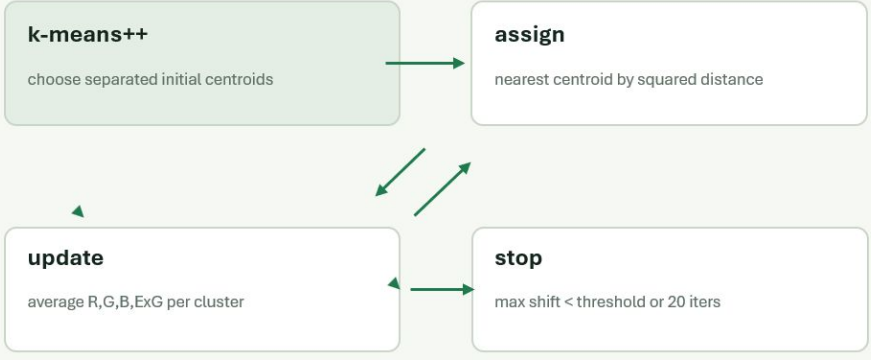
green: ExG >= T
stressed: $0.5T$ <= ExG < T
background: otherwise

K-means runs on a color + greenness feature vector

Each pixel is clustered using RGB plus Excess Green, so color and plant signal both matter.

[R, G, B, ExG]

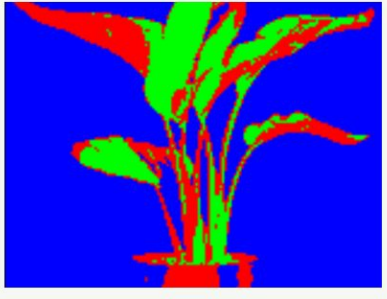
$$ExG = 2G - R - B$$



Adaptive thresholding

Otsu chooses an ExG split per frame; for the shown capture T ~= 49. This helps handle lighting changes and auto-levels.

EXG view



cluster view

Code refs: kmeans_init_plusplus(), kmeans_run(), otsu_threshold_exg()

Mask

Left: K-means Output

Pixel Cluster Map

0	0	0	2	2	3
0	0	2	2	2	3
0	1	1	2	3	3
1	1	1	2	3	3
1	1	2	2	3	3
1	1	2	2	3	3

Cluster Centroid Summary

Cluster	ExG_c	Green dominance?	Brightness	Role
0	high	yes	medium	GREEN
1	low	no	dark	BACKGROUND
2	medium	yes	not too bright	STRESSED
3	low	no	bright	BACKGROUND

Each pixel inherits a cluster_id from k-means in feature space [R, G, B, ExG].

Middle: Cluster Role Logic

Otsu-guided Role Assignment

Adaptive threshold T comes from Otsu on the ExG histogram.

$$\text{brightness}(c) = \frac{R_c + G_c + B_c}{3}$$

$$\text{green\_dom}(c) = G_c \geq 1.05 \times \max(R_c, B_c)$$

GREEN(c):
$\text{ExG}_c \geq T$ and green_dom(c)

STRESSED(c):
$0.5T \leq \text{ExG}_c < T$ and brightness(c) ≤ 160 and green_dom(c)

BACKGROUND(c):
otherwise

Right: Mask Formation and Cleanup

Cluster Map → Raw Mask → Clean Mask

Cluster Map:

0	0	0	2	2	3
0	0	2	2	2	3
0	1	1	2	3	3
1	1	1	2	3	3
1	1	2	2	3	3
1	1	2	2	3	3

Raw Mask:

1	1	1	1	0
1	1	1	1	0
1	1	1	0	0
0	0	0	1	0
0	0	1	1	0
0	0	1	1	0

Clean Mask:

1	1	1	1	0
1	1	1	1	0
1	1	1	0	0
1	1	1	1	0
0	0	1	1	0
0	0	1	0	0

Raw Mask is formed as:
mask[i] = GREEN(cluster_i)
OR STRESSED(cluster_i)

Largest 4-connected component

Meta Buffer Write

3	2	1	0
b_3	b_2	b_1	b_0

nibble[3:1] = cluster_id
nibble[0] = mask_bit

Displayed on VGA as cluster pane + mask pane.

Clusters become plant roles, then a clean plant mask

The software keeps the algorithm interpretable: cluster roles are chosen from centroid statistics

GREEN

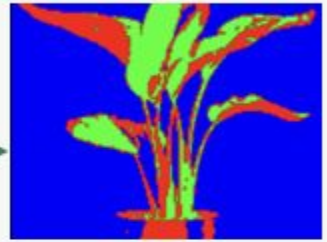
high ExG + green dominance

STRESSED

mid ExG, not too bright, still green-dominant

BACKGROUND

everything else



cluster view

mask view

Health features

plant_area

green_ratio

stressed_ratio

mean_exg

Rule-based verdict

No Plant -> Yellowing/Stressed -> Mild Stress -> Healthy / Mostly Healthy

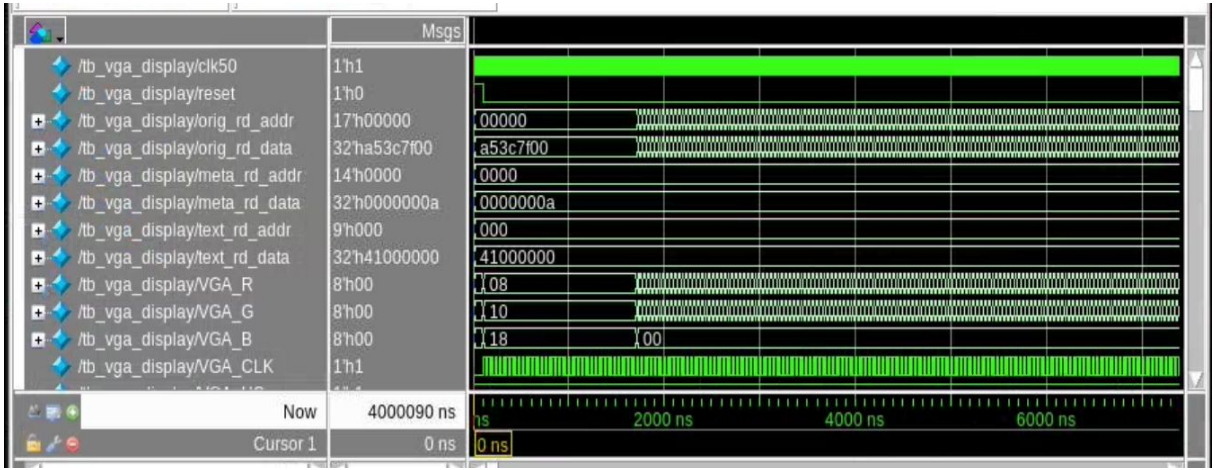
Priority is intentional: if stressed_ratio is high, that dominates the verdict; otherwise green coverage and mean ExG decide health.

Code refs: interpret_clusters(), compute_plant_mask_from_clusters(), largest_connected_component(), classify_plant_health()

Verification

Functional Verification in System Verilog

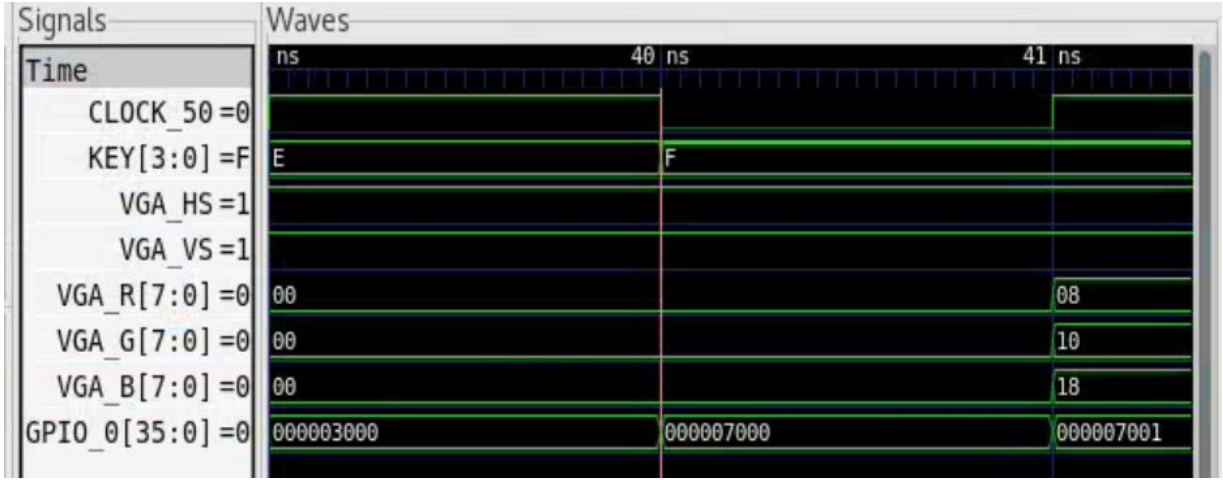
Test ID	Testcase Name	Where Implemented in TB	Objective / Feature Verified	Stimulus Applied	Expected Result / Checks
TC0	Power-On Reset / Basic Boot	All TBs	Verify reset initialization across modules	Reset asserted then deasserted	Registers reset, stable outputs, no X states
TC1	Buffer Write/Read Integrity	tb_buffer.sv	Verify synchronous RAM behavior	Sequential writes & reads	Correct data stored and retrieved
TC2	Control Register Latch + Ack Flow	tb_control_regs.sv	Verify Avalon write + ack behavior	avs_write + ack pulses	capture_start sets and clears correctly
TC3	Debounce Stability Filtering	tb_debounce.sv	Verify glitch rejection	Noisy input + stable press	Single clean pulse generated
TC4	Font ROM Lookup Correctness	tb_font.sv	Verify ROM character mapping	char_code + row scan	Correct 8×16 bitmap output
TC5	Frame Buffer Dual-Port Read/Write	tb_frame_buffer.sv	Verify camera + VGA/HPS access	Concurrent write/read	No corruption, correct latency
TC6	Meta Buffer Memory Consistency	tb_meta_buffer.sv	Verify AVS + VGA consistency	AVS write + VGA read	Data matches across interfaces
TC7	VGA Display Pixel Pipeline Test	tb_vga_display.sv	Verify full display pipeline	Synthetic memory inputs	HSYNC/VSYNC + pixel updates
TC8	Address Mapping Correctness	tb_vga_display.sv	Verify framebuffer addressing	Display scan region	Correct orig/meta/text mapping
TC9	VGA Timing Generation	tb_vga_display.sv	Verify sync generation	Continuous clock	Proper HSYNC/VSYNC timing
TC10	End-to-End System Integration	tb_vga_display.sv (extended)	Verify full system pipeline	Combined stimuli	Stable video output, no glitches



- This testbench verifies the VGA display pipeline, including memory address generation, multi-buffer data fetching, and final pixel rendering on VGA output.
- At the start, the system is held in reset to initialize all counters and internal registers. After reset deassertion, the display pipeline begins operation.
- We use three memory models: original image, meta buffer, and text buffer. Each of these is accessed using independent address generators, which can be observed as continuously changing address signals in the waveform.
- The RGB signals change dynamically, which confirms that pixel data is being correctly fetched from memory and rendered.
- Hence, this test validates the full VGA display architecture including multi-layer rendering and timing correctness

Verification in C++

Test ID	Testcase Name	Where Implemented in TB	Objective / Feature Verified	Stimulus Applied	Expected Result / Checks
TC0	Global Reset / Basic Boot	All TBs (system-wide)	Verify correct reset initialization across all RTL blocks	Reset asserted then deasserted	All modules initialized correctly, no X propagation
TC1	Buffer Memory Integrity (Single + Dual Clock + Latency)	tb_buffer.cpp	Verify synchronous RAM behavior, clock-domain separation, and read stability	Sequential writes + independent wr_clk/rd_clk toggling	Data integrity maintained, no race conditions, correct readback
TC2	Control Register Interface	tb_control_regs.cpp	Verify Avalon register write/read + control handshake logic	AVS write transactions + capture start trigger	Correct register updates and ack behavior
TC3	Debounce Logic Validation	tb_debounce.cpp	Verify switch debounce and glitch filtering	Noisy input followed by stable press	Single clean pulse generated per press
TC4	Font ROM Functionality	tb_font.cpp	Verify ROM lookup correctness for character generation	Sweep ASCII (A/0/Z) across rows	Stable, correct bitmap output with no X values
TC5	Frame Buffer System (AVS + VGA + Camera + Frame Handling)	tb_frame_buffer.cpp	Verify dual-port memory, latency, frame sync, and HPS access	Write/read via AVS + VGA read + frame boundary updates	Correct data consistency, proper frame reset, no tearing
TC6	Meta Buffer Shared Memory Access	tb_meta_buffer.cpp	Verify shared memory correctness between AVS and VGA	AVS write + VGA read access	Consistent data across both interfaces
TC7	VGA Display Pipeline & Timing	tb_vga_display.cpp	Verify full VGA pipeline (timing, address gen, RGB output, long-run stability)	Continuous clk50 run (~200k cycles)	Stable HSYNC/VSYNC, valid RGB transitions, no deadlock
TC8	End-to-End System Integration	system TB (camera → frame → VGA)	Verify full system pipeline integration	Simulated OV7670 + frame flow	Continuous correct VGA output updates



HS toggles: 374 : Horizontal scanning active
Pixel changes: 6419 : Active pixel data generation verified
RESULT: FULL SYSTEM VGA TEST PASSED

The waveform demonstrates correct VGA synchronization and active pixel generation with continuous HSYNC toggling and dynamic RGB output driven by simulated camera input.

Conclusion

- OV7670 works

- Auto Level helps, a good image is created

- Computer Vision is hard

References

1. S. Sankaran, A. Mishra, R. Ehsani, and C. Davis, “A review of advanced techniques for detecting plant diseases,” *Computers and Electronics in Agriculture*, vol. 72, no. 1, pp. 1–13, 2010.
2. D. M. Woebbecke, G. E. Meyer, K. Von Bargen, and D. A. Mortensen, “Color indices for weed identification under various soil, residue, and lighting conditions,” *Transactions of the ASAE*, vol. 38, no. 1, pp. 259–269, 1995.
3. G. E. Meyer and J. C. Neto, “Verification of color vegetation indices for automated crop imaging applications,” *Computers and Electronics in Agriculture*, vol. 63, no. 2, pp. 282–293, 2008.
4. Donald G. Bailey, *Design for Embedded Image Processing on FPGAs*, Wiley, 2011.
5. J. MacQueen, “Some methods for classification and analysis of multivariate observations,” *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.
6. OmniVision Technologies, *OV7670 CameraChip Sensor Datasheet*, 2006.
7. Intel Corporation, *Cyclone V Device Handbook*, 2018.
8. J. Cho, M. Mirzaei, J. Oberg, and R. Kastner, “FPGA-based real-time embedded system for image processing applications,” *EURASIP Journal on Embedded Systems*, 2009.