CSEE4840 Design Document

Linus Lei, Daolin Li, Gurleen Kalra

Introduction

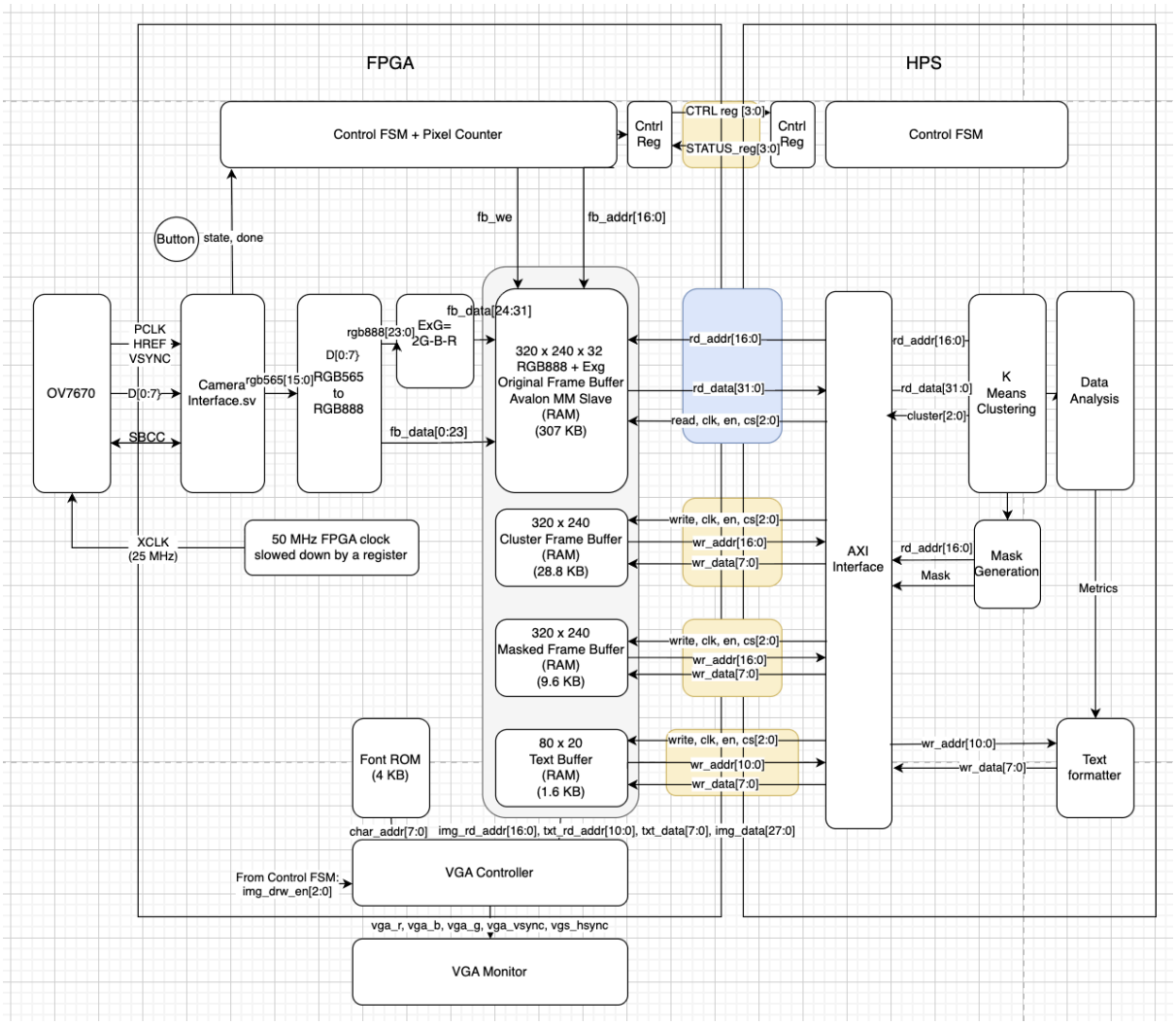
This project implements an embedded plant-monitoring system on the DE1-SoC platform using a combination of FPGA hardware and HPS software. An OV7670 camera is used to capture an image of a plant, and the resulting image is processed to estimate plant condition from its color and greenness characteristics. The system is designed to be lightweight, interpretable, and suitable for real-time embedded operation, with the FPGA handling timing-critical tasks and the HPS performing higher-level image analysis.

On the hardware side, the FPGA interfaces directly with the OV7670 camera, generates the required control clocks, receives the RGB565 pixel stream, and converts it into RGB888 format. In parallel, it computes an additional vegetation-related feature, Excess Green (ExG), for each pixel. The FPGA then packs the RGB and ExG values into a 32-bit word and stores the captured frame in on-chip memory. The same FPGA fabric also manages the VGA output path, reading from image and text buffers to produce the final display.

On the software side, the HPS reads the captured frame through the HPS-to-FPGA bridge and performs k-means clustering on the pixel data. The clustering result is used to separate plant regions from background and to distinguish healthier green regions from more stressed or yellow regions. From this result, the system generates a plant mask, computes simple health-related statistics, and produces a rule-based summary of plant condition. These processed outputs are then written back to FPGA-resident buffers so they can be displayed alongside the original image.

The final system provides both visual and textual feedback through a VGA monitor. The top portion of the screen displays the original image together with processed views, while the lower portion shows text-based results and status information. Overall, this project demonstrates a clear hardware–software partition: the FPGA performs deterministic image capture and display tasks, while the HPS carries out the more flexible image-processing and decision-making algorithms.

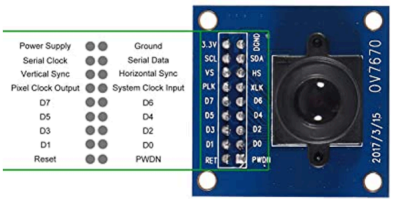
Block Diagram



Interfacing OV7670

To capture the plant image, an OV7670 camera is used. It's a low-power compact image sensor which can be configured to output up to 640 x 480 VGA video stream at 30 frame per second. The OV7670 can be configured via the SCCB interface by Omnivision to adjust its output format, white balance and so on. The Pinout of OV7670 is included below, it utilize an 8-bit parallel output alongside HREF and VSYNC for locating the line and frame. It also needs a XCLK as input clock and regulates its 8-bit output via PCLK. 3.3V powering is needed.

OV7670 Camera Pinout



The SCCB interface is compatible with I2C protocol. Primarily we would use a 3-phase write procedure to write to the configuration registers of the camera. The SCL would be provided and set to 100KHz by the FPGA per the datasheet. It would be generated using a counter to slow down the 50 MHz FPGA clock.

The XCLK would be provided from a register which halves the 50 MHz FPGA clock to 25 MHz, suitable for the camera per its datasheet.

Our design will also utilize a shutter button that captures a single frame. This button will be debounced. Once triggered, it looks the next VSYNC to capture the next complete frame of the camera.

By properly configuring OV7670, a QVGA output with resolution 320 x 240 will be outputted in the format of RGB565, with each pixel taking 2 clock, 16 bits to transmit.

We will implement three modules:

1. SCCB module for configuration
2. RGB565 Receive Module that identifies VSYNC and HREF
3. 25 MHz XCLK generator

Processing (FPGA side):

Once the RGB565 datastream is received, it will go through two steps of processing. RGB565 to RGB888 and ExG calculation.

The first step of processing is to isolate the RGB component from the 16 bit input then each RGB component to 8 bit, thus transforming to RGB888.

ExG is a metric to measure how green a specific pixel is. It's more accurate than just the green component itself since it considers the green component with respect to red and blue. The exact formula is simple: ExG = 2G - B - R. It's simple to implement and fast to process. ExG value will also be formatted into 8 bits. After getting ExG, it will be packed together with RGB888 as a 32 bit word into the main frame buffer.

Frame Buffer

The system uses several FPGA-resident frame buffers to separate image acquisition, software analysis, and video display. The main image buffer is the Original Frame Buffer, which stores one captured camera frame at a resolution of 320 × 240. Each pixel is stored as a 32-bit word. The RGB data is expanded from the OV7670's RGB565 format to RGB888 in FPGA logic before being written into the buffer. The buffer therefore contains 76,800 words and occupies about 307 KB of on-chip memory. Pixels are stored in row-major order, so the address of pixel (x, y) is computed as y * 320 + x.

The Original Frame Buffer is written by the FPGA camera pipeline and exposed to the HPS as a read-only Avalon-MM slave. Internally, the FPGA control FSM and pixel

counter generate the write address and write enable signals during frame capture. On the software side, the HPS reads the buffer through the HPS-to-FPGA bridge to obtain the image for k-means clustering and analysis. A local read path is also provided to the VGA display pipeline so the original image can be displayed without copying it into software memory.

In addition to the Original Frame Buffer, the design includes additional display buffers: Cluster & Mask Buffer. These buffers are written by the HPS after image processing is complete. The Cluster Buffer stores the clustering result for each pixel, while the Mask Buffer stores the plant segmentation result. Both are read by the VGA display pipeline to generate the processed output views. A separate Text Buffer stores ASCII character codes for the on-screen status display. Unlike the image buffers, the text itself is rendered in hardware using a font ROM inside the FPGA.

Hardware - Software Interface

The HPS accesses the FPGA through the Cyclone V HPS-to-FPGA bridges, while the FPGA-side resources are exposed as memory-mapped slave peripherals. Functionally, the HPS acts as the bus master and the FPGA buffers and control registers act as bus slaves. Cyclone V provides both a standard HPS-to-FPGA bridge and a lightweight HPS-to-FPGA bridge, and the lightweight bridge.

The interface consists of two categories:

1.Control registers, used for synchronization between capture logic and software

2.Memory-mapped buffers, used for bulk data exchange

The bridge assignment follows this division. The lightweight HPS-to-FPGA bridge is used for the small control-register block, since it carries only low-bandwidth control and status information. The full HPS-to-FPGA bridge is used for the larger data-bearing slaves: the Original Frame Buffer, Cluster & Mask Buffer, and Text Buffer.

Control Registers

The control-register block is implemented on the FPGA side as a small 32-bit memory-mapped slave connected to the lightweight HPS-to-FPGA bridge.
It contains two registers: CONTROL and STATUS. These registers are the handshake mechanism between the FPGA control FSM and the HPS control software.

Register Map

CONTROL[2]	display_enable
CONTROL[1]	clear_frame_valid
CONTROL[0]	capture_start
STATUS[2]	error
STATUS[1]	frame_valid
STATUS[0]	capture_busy

The control interaction proceeds as follows:

1.The HPS control software writes CONTROL[0] = 1 to start capture.

2.The FPGA Control FSM enters the capture state and sets STATUS[0] = 1 (capture_busy).

3.The FPGA camera pipeline fills the Original Frame Buffer using fb_we, fb_addr[16:0], and fb_data[31:0].

4.When the last pixel is written, the FPGA clears STATUS[0] and sets STATUS[1] = 1 (frame_valid).

5.The HPS polls STATUS[1] until it becomes 1.

6.The HPS reads the Original Frame Buffer and performs K-means clustering, mask generation, and data analysis.

7.After processing, the HPS writes the derived buffers and optionally updates display state.The HPS writes CONTROL[1] = 1 to clear the valid flag and allow the next capture cycle.

Original Frame Buffer

It stores one frame at resolution 320 × 240, with each pixel occupying 32 bits. This buffer is connected to the full HPS-to-FPGA bridge.The 32-bit word is packed as:

fb_word[31:24] = ExG

fb_word[23:16] = Red

fb_word[15:8] = Green

fb_word[7:0] = Blue

The frame contains: 320 × 240 = 76,800 pixels. Each pixel occupies one 32-bit word, so the buffer depth is 76,800 words. The address therefore uses 17 bits: fb_addr[16:0]. The address is row-major: addr = y * 320 + x.

Cluster & Masked Frame Buffer

To reduce FPGA memory usage, the clustering result and plant mask are merged into a single packed metadata buffer. Each pixel is represented by 4 bits, where bits [3:1] store the 3-bit cluster ID and bit [0] stores the binary plant mask. Eight pixels are packed into one 32-bit word, reducing the total storage requirement for a 320×240 frame to 9,600 words (38.4 KB). The HPS writes this packed metadata buffer after processing, and the VGA controller reads and unpacks the metadata to generate the clustered and masked display views.

The combined metadata buffer is organized as a 32-bit word array, where each word stores the cluster and mask information for 8 consecutive pixels. For pixel (x, y), the linear pixel index is first computed as p = y * 320 + x. The corresponding word address is then word_idx = p >> 3, since each 32-bit word contains 8 pixels, and the pixel’s position within

that word is slot = p & 7. Each pixel occupies one 4-bit nibble, where bits [3:1] store the 3-bit cluster ID and bit [0] stores the 1-bit mask. Therefore, the bit offset of that pixel inside the 32-bit word is shift = slot * 4, and the metadata can be extracted as meta = (word >> shift) & 4'hF.

Text Buffer

To reduce hardware complexity while still supporting flexible on-screen messaging, the text display is implemented using a dedicated text buffer and a font ROM on the FPGA side. The text buffer stores ASCII character codes rather than pre-rendered pixel values, allowing the HPS to update displayed text by writing only character data. Each character is represented by 8 bits, and the text region is organized as an 80 × 20 character grid, requiring a total of 1,600 bytes of storage. To match the 32-bit memory interface, four characters are packed into one 32-bit word, reducing the total storage requirement to 400 words (1.6 KB). The HPS writes this packed text buffer after formatting output strings such as status messages and analysis results, and the VGA controller reads the stored ASCII codes and uses the font ROM to render the final text on screen.

The text buffer is organized as a 32-bit word array, where each word stores the ASCII codes for 4 consecutive characters. For character position (row, col), the linear character index is first computed as c = row * 80 + col. The corresponding word address is then word_idx = c >> 2, since each 32-bit word contains 4 characters, and the character's position within that word is slot = c & 3. Each character occupies one 8-bit byte, so the bit offset of that character inside the 32-bit word is shift = slot * 8, and the ASCII code can be extracted as char_code = (word >> shift) & 8'hFF. The VGA controller uses this character code as the

address input to the font ROM, then combines the glyph bitmap with the current pixel position to render the appropriate text pixels in the display region.

Processing (HPS side)

The HPS is responsible for the software-side analysis of the captured plant image. After the camera frame is obtained, the HPS converts the image into a usable RGB representation and performs a complete plant-health analysis pipeline.

The main software-side algorithm in this project is k-means clustering, which groups pixels with similar color and greenness properties. Each pixel is represented as a four-dimensional feature vector:

$$x_i = [R_i, G_i, B_i, ExG_i]$$

where R_i, G_i, and B_i are the RGB888 components and ExG_i is the Excess Green value computed in hardware.

The algorithm maintains K centroids, where centroid j is:

$$\mu_j = [\mu_{j,R}, \mu_{j,G}, \mu_{j,B}, \mu_{j,ExG}]$$

For each pixel, the HPS computes the squared Euclidean distance to every centroid:

$$d_{ij} = (R_i - \mu_{j,R})^2 + (G_i - \mu_{j,G})^2 + (B_i - \mu_{j,B})^2 + (ExG_i - mu_{j,ExG})^2$$

The pixel is assigned to the cluster with minimum distance:

$$c_i = argmin_j d_{ij}$$

After all pixels have been assigned, each centroid is updated by averaging the feature vectors of all pixels assigned to that cluster. If S_j is the set of pixels currently belonging to cluster j, then the new centroid is:

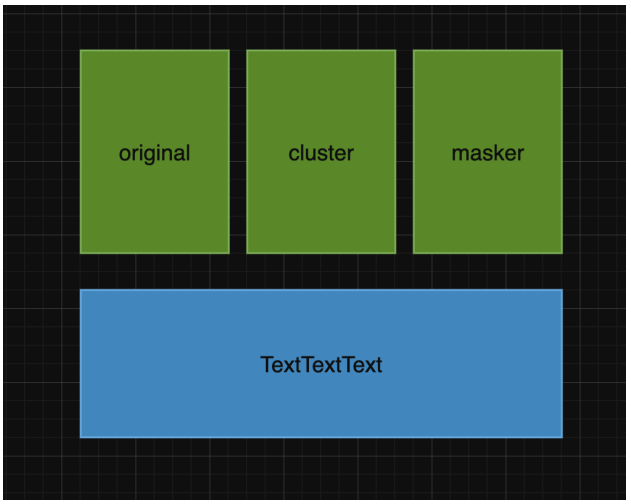
$$\mu_j = (1/|S_j|) \sum_{iinS_j} x_i$$

These assignment and update steps are repeated until convergence or until a fixed maximum number of iterations is reached.

After segmentation, the HPS extracts several health-related features, including the proportion of green pixels, the proportion of stressed or yellow pixels, the mean ExG value, and the detected plant area. Finally, these features are evaluated by a rule-based classifier to determine the overall plant condition, such as Healthy, Mostly Healthy, Mild Stress, Yellowing, Stressed, or No Plant. This processing flow allows the system to provide an interpretable and lightweight health assessment suitable for embedded plant-monitoring applications.

VGA Display

The VGA display is divided into two main regions: an upper image region and a lower text region. The upper region shows the original, cluster, and masked images. Since the stored frame size is 320×240, each image is downsampled by 2 before display, so each pane appears as 160×120 on the VGA screen. Placing three such panes side by side uses 480×120 pixels total. The remaining horizontal space on the 640-pixel-wide screen is left blank to create visible separation between the three image windows.



The lower part of the screen is used for text output. It is implemented as an 80-column by 20-line text region. With an 8×16 font, this exactly occupies 640×320 pixels. A blank gap is left between the top image region and the lower text region so that the screen is visually separated into analysis images above and status/output text below.

This layout is generated entirely by the FPGA VGA controller. For each screen pixel (x, y), the controller first determines which display region the pixel belongs to. If the pixel is inside one of the three image windows, the controller computes the corresponding source address in the stored frame buffers and reads the appropriate data. If the pixel is in the text region, the controller reads the character code from the text buffer and the glyph pattern from

the font ROM, then renders the text pixel. Pixels outside these defined regions are assigned a fixed background color, producing the blank spaces in the final display.

Resource Budget

FPGA buffer: 320 x 240 x 4 (bytes per pixel) = 307.2 KB (RGB888: 24bits, ExG: 8 bits)

Cluster +Masked Buffer: 320 x 240 x 4bit = 38.4 KB

Font Memory: 8 x 16 x 256 (characters) = 4 KB

Text Input Memory = 20 lines x 80 columns x 8 bit per character = 1.6 KB

Summary: 307.2 + 38.4 + 4 + 1.6 = 351.2 KB