Real-Time Image Processing with Median Filtering

This project focuses on real-time image processing using a median filter to enhance image quality. By integrating an external camera module, the system captures video streams, processes the data using FPGA-based logic, and displays the filtered output via the board's VGA interface. The design ensures efficient and accurate processing for live image enhancement

By : Dror Heller , Yahel Yaish , Chaya Rotenberg .

Block Diagram and Descriptions

Camera OV7670

- Captures live video frames for processing.

Write Manager

- Manages data writing to the line buffer.

Line Buffer

- Temporarily stores pixel data for Processing

Read Manager

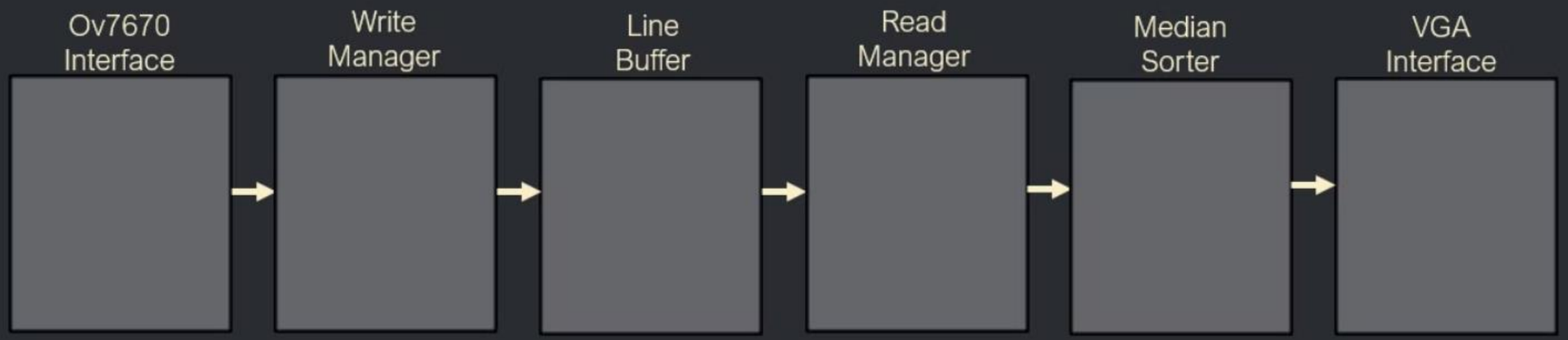
- Handles data retrieval from the line buffer.

Median Sorter

- Processes pixel data using a median filter .

VGA Interface

- Displays the processed video output on the screen.



Camera OV7670:

Captures live video frames and streams pixel data synchronized with clock signals.

Input Ports

- **clk100:** Primary clock input at 100MHz for system synchronization.
- **OV7670_VSYNC:** Vertical synchronization signal, indicating the start of a new frame.
- **OV7670_HREF:** Signal indicating the active pixel region of the current row.
- **OV7670_PCLK:** Pixel clock signal for synchronizing pixel data.
- **OV7670_D:** 12-bit data bus input for pixel data from the camera.

Output Ports

- **OV7670_SIOC:** Output clock signal for I2C communication to configure the camera.
- **OV7670_RESET:** Output signal to reset the camera module.
- **OV7670_PWDN:** Output signal to enable or disable the camera for power saving.
- **OV7670_XCLK:** External clock output sent to the camera for its operation.

Bi-Directional Ports

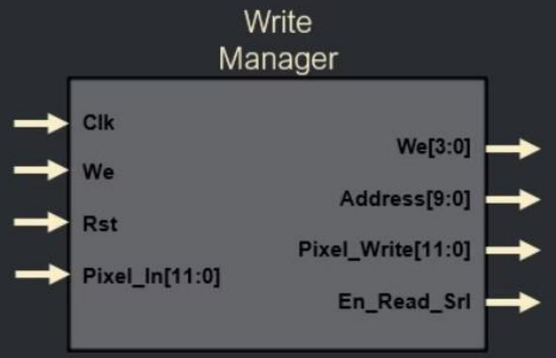
- **OV7670_SIOD:** Bi-directional data line for I2C communication to send or receive configuration data.

Write Manager:

Manages pixel data flow by writing it sequentially to the appropriate line buffer for further processing

Input Ports

- **clk:** Clock signal for synchronizing operations within the write controller.
- **we:** Write enable signal to control data writing into the buffer.
- **pixel_in:** 12-bit input representing pixel data to be written into the line buffer.
- **rst:** Reset signal to initialize the internal counters and states.



Output Ports

- **we_1→4:** Write enables signals for selecting the appropriate RAM (line buffer). Each signal corresponds to one of the four RAM blocks.
- **address:** 10-bit output indicating the current memory address for writing data (supports 640 columns).
- **pixel_w:** 12-bit pixel data output passed to the line buffer.
- **enable_read_srl:** Signal to enable reading from the shift register logic (SRL) when ready.

Line Buffer:

A single-row memory implemented as BRAM with 640 addresses, each storing 12 bits of pixel data. This buffer temporarily holds one row of the image, allowing efficient access and sequential processing for image filtering operations.

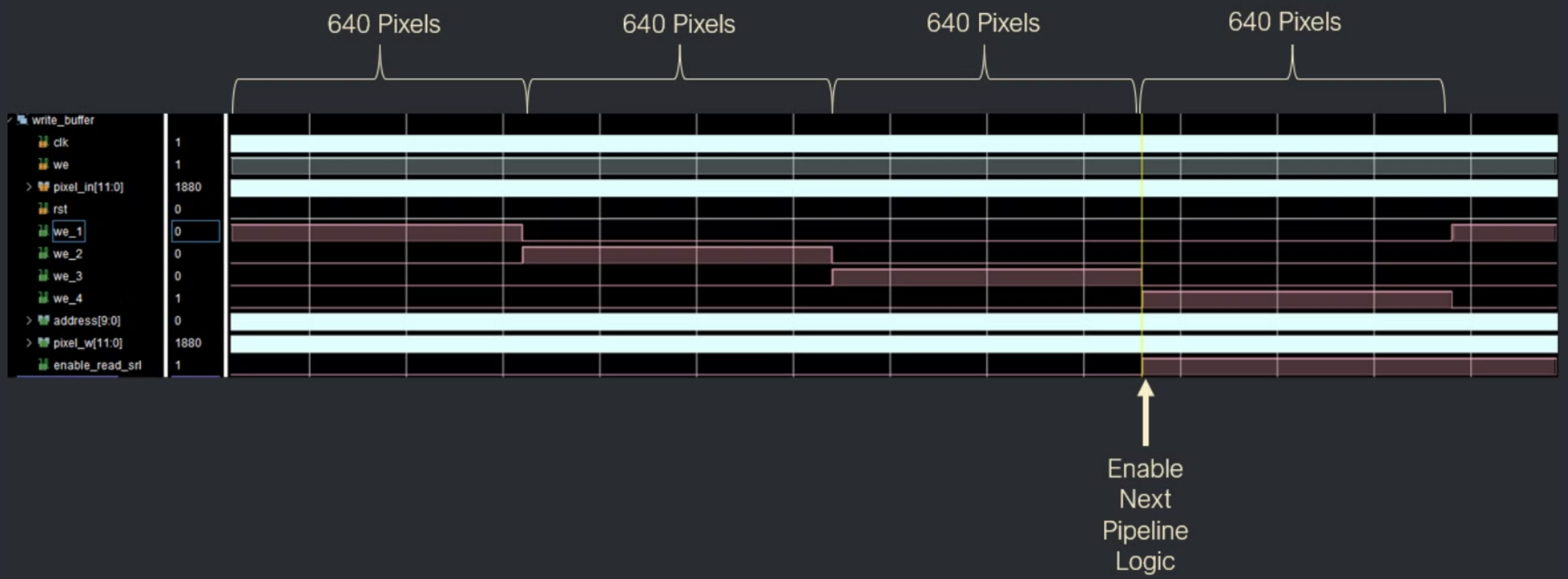
Input Ports

- **clk:** Clock signal for synchronizing read and write operations in the BRAM.
- **we:** Write enable signal to control whether data is written to the buffer.
- **address:** 10-bit input specifying the memory address (column index) for the read/write operation (0–639).
- **data_in:** 12-bit input representing the pixel data to be written into the buffer.

Output Ports

- **data_out:** 12-bit output providing the pixel data read from the buffer at the specified address.

Write Manager to Line Buffer Simulation:

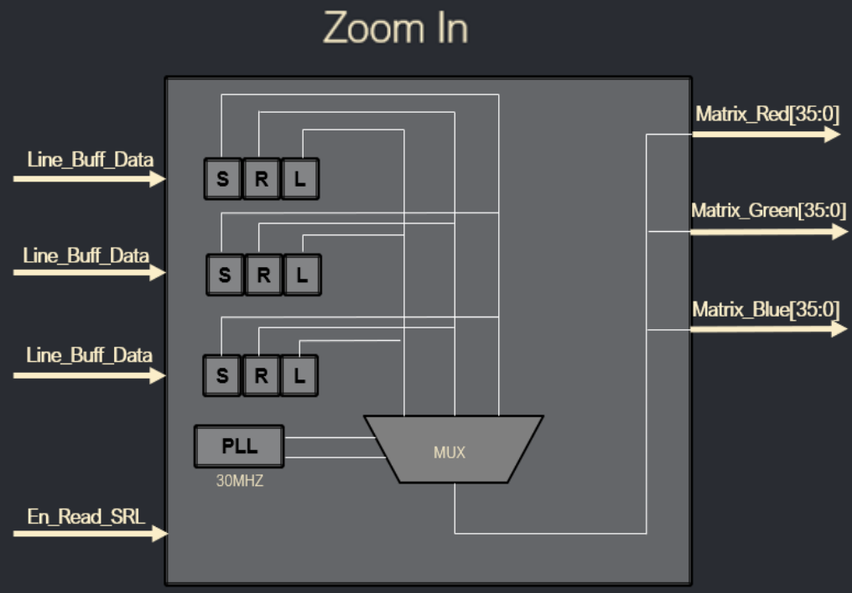


Read Manager:

Manages fast pixel data retrieval from the line buffers, using SRLs and a 30MHz multiplexer to read three items within the same period as one incoming pixel at 10MHz

Input Ports

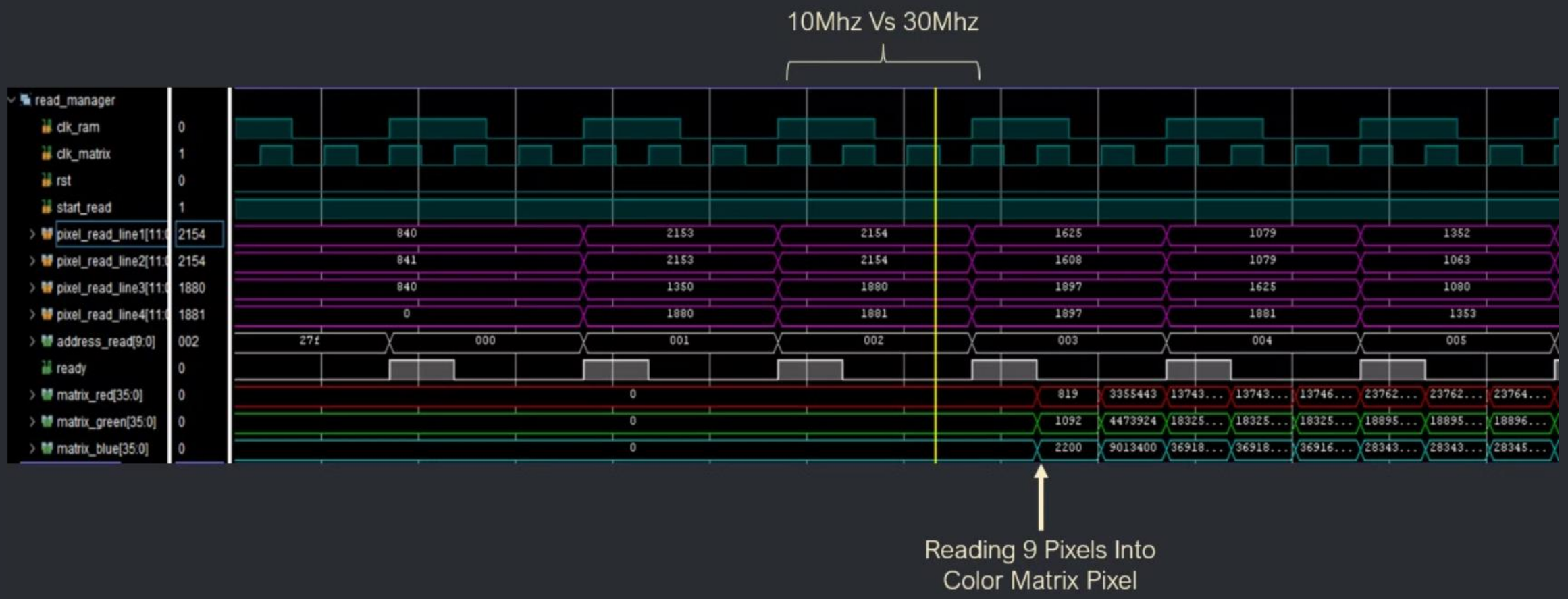
- **clk_ram:** Clock signal for accessing line buffers at 10MHz.
- **clk_matrix:** Clock signal for SRL and multiplexer operation at 30MHz.
- **rst:** Reset signal to initialize counters and internal states.
- **start_read:** Signal to start the reading process.
- **pixel_read_line1→4:** 12-bit inputs representing pixel data from four-line buffers.



Output Ports

- **address_read:** 10-bit output indicating the address of the line buffer being read.
- **ready:** Signal indicating when the matrix is ready for the next stage of processing.
- **matrix_red, matrix_blue, matrix_green:** Outputs containing 36-bit data for each color channel (R, G, B) in a 3x3 matrix format for further processing.

Read Manager Simulation :



Median Sorter:

Performs a sorting operation to compute the median of a 3x3 pixel matrix, utilizing an even-odd sorting algorithm for efficient processing.

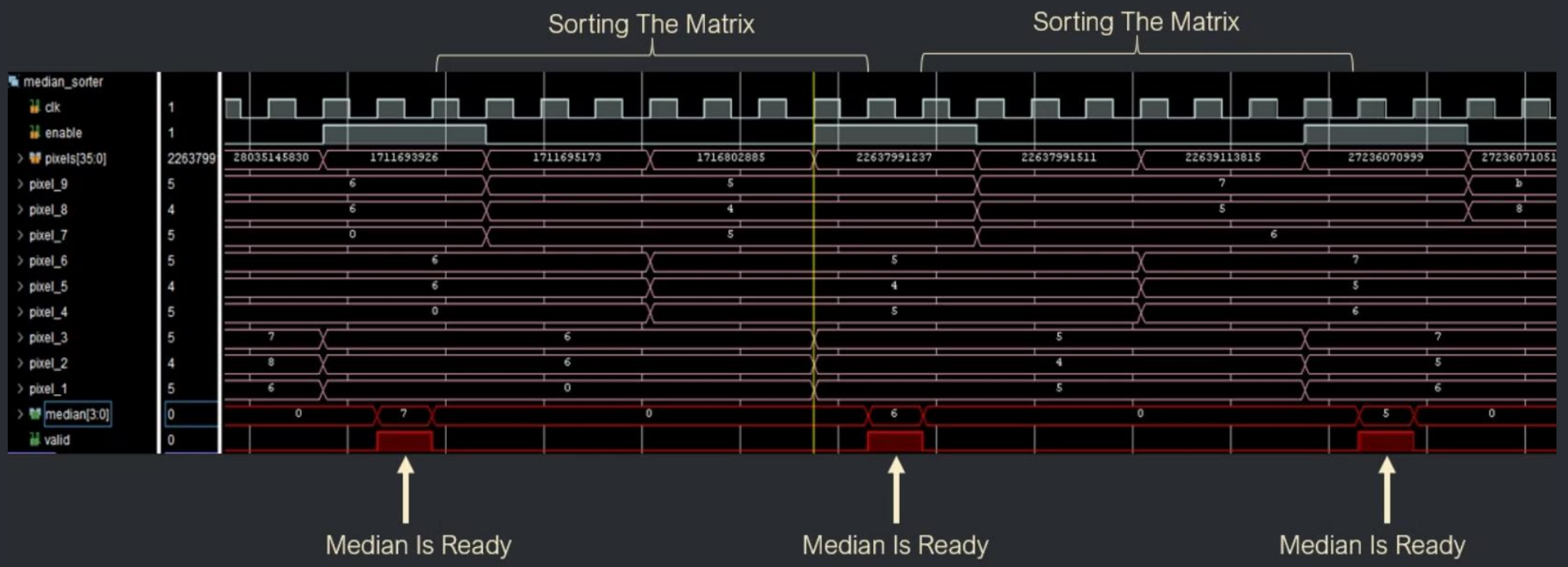
Input Ports

- **clk:** Clock signal for synchronizing operations.
- **enable:** Signal to enable the sorting process.
- **pixels:** Input vector containing 9 pixels (each 4 bits) representing the 3x3 matrix.

Output Ports

- **median:** 4-bit output representing the median value of the sorted pixel matrix.
- **valid:** Signal indicating when the median calculation is complete and the output is valid

Simulation for Median sorter



VGA Interface

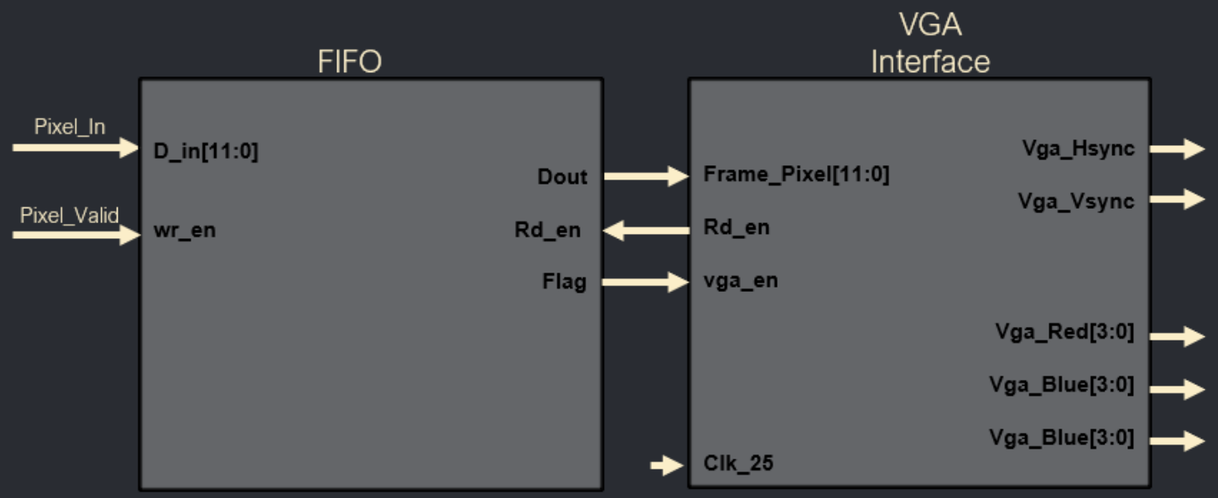
Handles the display of processed video output by generating VGA signals and mapping pixel data to the screen.

Input Ports

- **clk25:** 25MHz clock signal required for VGA timing and synchronization.
- **frame_pixel:** 12-bit input representing pixel data (RGB) to be displayed.

Output Ports

- **vga_red/green/blue:** 4-bit outputs for each color channel (R, G, B), controlling pixel color intensity on the screen.
- **vga_hsync:** Horizontal synchronization signal for controlling row timing on the VGA display.
- **vga_vsync:** Vertical synchronization signal for controlling frame timing on the VGA display.
- **frame_addr:** 19-bit output indicating the current frame memory address for fetching pixel data



Filtering Results: Noise Reduction Demonstrated

The results highlight the effectiveness of our median filter in reducing noise and improving image quality. Below, you can see a comparison between the original noisy image and the cleaned output after applying the filter

Challenges and Solutions

During the project, we encountered several challenges that required innovative solutions. Below are the key issues and how we addressed them:

1) Providing Multiple Clock Signals :

Problem

We needed to supply different clock frequencies to various components to ensure proper synchronization.

Solution

Utilized a MMCM to generate multiple clock signals efficiently.

---

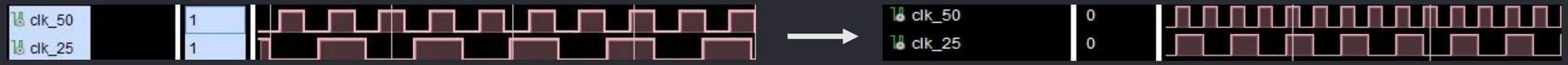
2) Overloading the MMCM :

Problem

Overloading the MMCM with too many clock outputs caused it to fail to lock properly, resulting in phase mismatches and clock skew between components.

Solution

Reduced one clock output from the MMCM and replaced it with a BUFR, restoring functionality and saving power, as a MMCM consumes approximately 125mW.



Challenges and Solutions

3) Camera Module Configuration:

Problem

Difficulty in understanding the camera’s input and output ports, connections, and configuration, as we adapted an existing project.

Solution

Analyzed the camera documentation and project structure to ensure proper synchronization and port mapping.

---

4) Understanding VGA Protocol:

Problem

Challenges in working with VSYNC and HREF signals and understanding VGA timing standards.

Solution

Studied VGA protocol specifications and adjusted signal generation to align with timing requirements

---

5) VGA Timing and Data Synchronization:

Problem

Pixel data was received at 10MHz but needed to be sent to the VGA at 25MHz, requiring synchronization between the different clock domains.

Solution

Implemented a FIFO buffer to handle data transfer between the two clock rates. Carefully balanced read and write operations to ensure valid data was written and read correctly.

Project Timeline and Milestones

Project Initiation

- Defining project goals and requirements.
- Selecting hardware components

Implementation Phase

- Integrating individual modules (camera, VGA, memory buffers).
- Testing clock synchronization and data flow.

Final Integration

- Combining all components into a cohesive system.
- Performing real-time tests with live data.

1

2

Design Phase

- Developing system architecture and block diagrams.
- Writing initial VHDL code for core components.

3

4

Testing and Debugging

- Verifying functionality through simulations.
- Resolving clock skew and timing issues.

5

Insights and Future Enhancements

Throughout the project, we gained valuable insights into hardware design, synchronization challenges, and real-time image processing. For future enhancements, the following features could be integrated to improve the system's functionality and user interaction:

1. **Mouse Interface**
   - Allows users to interact with the image, such as coloring specific pixels or selecting regions for zooming.
   - Enhances usability by providing a direct method for manipulating the visual output.
2. **Bluetooth Module**
   - Enables command transmission from a mobile app to the FPGA.
   - Adds flexibility and functionality by allowing remote control and wireless communication.
3. **Keyboard Interface**
   - Provides a way for users to input commands, annotate images, or trigger specific processing tasks.
   - Improves accessibility and expands the system's use cases for advanced applications.

By incorporating these features, the system would become more interactive, versatile, and user-friendly, paving the way for broader adoption and innovative use cases.