Faculty of Engineering

Department of Computer Science & Engineering

Image Processing Accelerator using FPGA

Hardware Design Project Report

CS4363- Hardware Description Languages

200444R- P. D. N. T. Paramulla

200731U- W. A. S. P. Wijesuriya

17 December 2024

Table of Content

Description of the project idea

The objective of this project was to design, implement, and test an Image Processing Accelerator using an FPGA. The accelerator performs essential image processing tasks such as image filtering and edge detection, leveraging the parallel processing capabilities of FPGA hardware.

The system operates by preloading an image into the FPGA's Random Access Memory (RAM). This image is then processed and displayed on a monitor via the FPGA's VGA output interface. The real-time interactivity of the system is achieved through the use of on-board switches. By toggling these switches, users can apply various linear filters to the image dynamically, with the results reflected on the monitor in real time.

This project demonstrates the potential of FPGA-based hardware acceleration for computationally intensive image processing applications, providing high-speed processing and low latency compared to software-based implementations.

Development methodology, tools used and testing strategy

Development Methodology

The development of the Image Processing Accelerator follows a structured methodology, leveraging the parallel processing capabilities of the FPGA. The primary components of the system include a True Dual Port RAM, image preprocessing modules, and a VGA controller for image display.

1. **Memory Initialization**:
   A True Dual Port RAM is instantiated on the FPGA board. During project initialization, an image is loaded into the RAM from an initialization (.coe) file. This image serves as the base input for the image processing tasks.
2. **Parallel Module Execution**:
   Two distinct modules operate in parallel, each utilizing one of the two ports of the Block RAM. Each module runs on a separate clock input, ensuring efficient parallel processing.
3. **Image Filtering**:
   Several 3x3 filter kernels are stored in the FPGA’s ROM. Upon board initialization, the user can select a specific filter kernel using on-board switches. The selected kernel is then applied to the image stored in the RAM. The image processing is performed pixel-by-pixel.
    - **Pixel Selection and Kernel Application**:
      For each target pixel to be processed, the values of the 8 neighboring pixels are loaded into a processing module. This module applies the selected 3x3 filter kernel to the 9-pixel matrix, producing a filtered pixel value. The processed pixel is then written to a separate region of the Block RAM via Port A.

4. **VGA Controller**:
   The second module operates as a VGA controller. It reads the processed image data from Port B of the Block RAM and transmits it to a connected monitor. This module generates a 4-bit grayscale VGA signal to display the image in black and white. By doing so, the processed image is presented on the monitor in real time, reflecting the results of the selected image filter.

This methodology ensures real-time, hardware-accelerated image processing while maintaining a clear separation of functional responsibilities for modular and efficient design.

Tools Used

- Xilinx Vivado 18.2
- DIGILENT NEXYS A7 - 100T FPGA board

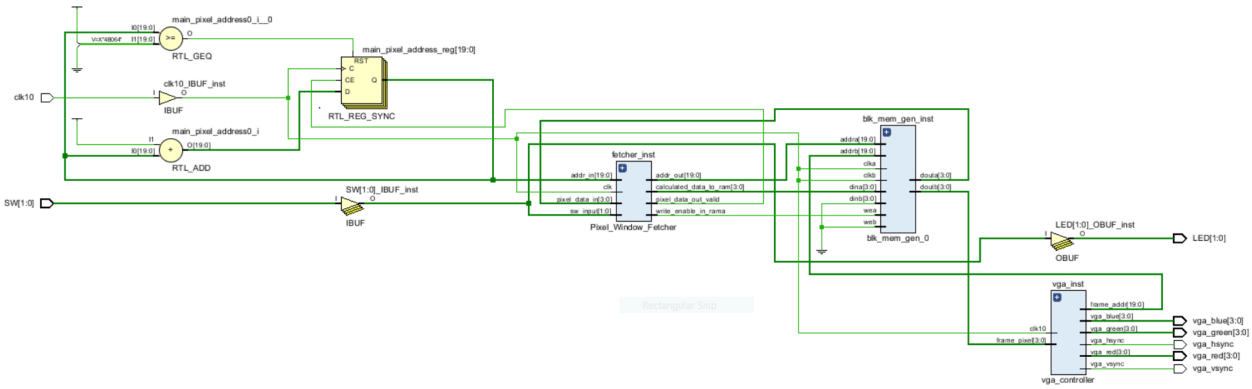
Testing Strategy

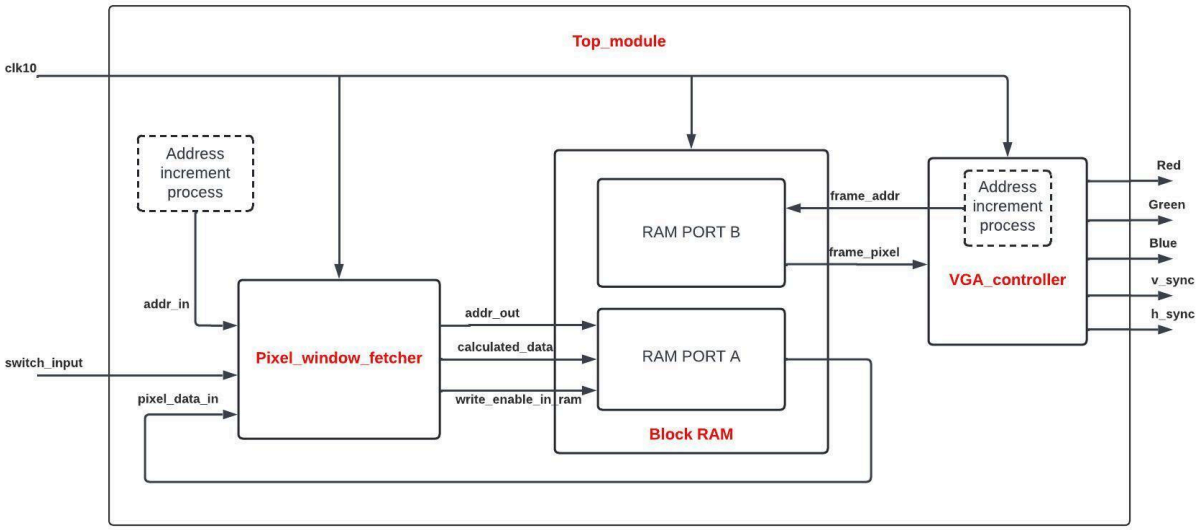
To ensure the functionality and correctness of the Image Processing Accelerator, a comprehensive testing strategy was adopted. This strategy focused on verifying the behavior of both the VGA controller and the image processing module through simulation and internal signal monitoring.

1. **Testbench Development**:
   A dedicated testbench (TB) file was created to test the functionality of the VGA controller and the image processing module. The testbench simulates the system's operation under controlled input conditions, ensuring each module responds as expected. The testbench validates pixel processing logic and image display functionality.
2. **Input Control and Signal Monitoring**:
   The only inputs provided by the testbench were the clock signals and on-board switch signals. Internal signals and VGA output signals were monitored to observe the system's behavior. By focusing on these key signals, the testbench verified the system's state transitions and the correctness of image processing outputs.
3. **Behavioral Simulation**:
   The output and internal signals of the top-level (TOP) module were analyzed using behavioral simulation. Key signals, such as pixel addresses, processed pixel data, and control logic, were monitored. This approach facilitated early detection and correction of logic errors, ensuring that the system met its functional requirements before hardware synthesis.

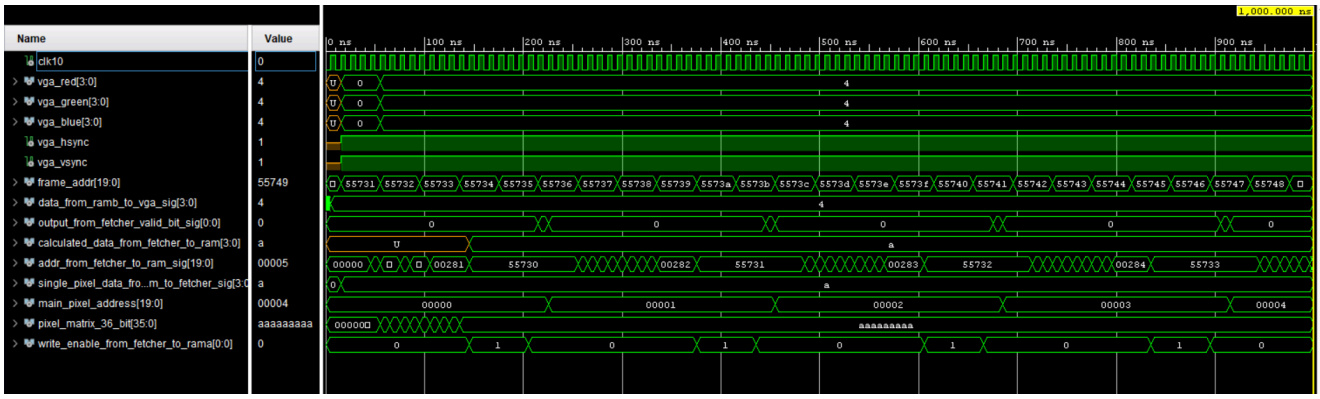
By combining testbench validation, controlled input testing, and behavioral simulation, the testing strategy provides robust verification of the system's functionality and supports a smooth transition to hardware implementation.

Overall Design / Modular architecture of the system





Simulations and results



Hardware implementation and results including Input/Output mapping

Input/Output Mapping

The project was implemented on a Nexys A7-100T FPGA board. Key specifications of the board that were utilized in the project are as follows:

- **15,850 Programmable Logic Slices**: Each slice contains four 6-input Look-Up Tables (LUTs) and 8 flip-flops.
- **100 MHz Inbuilt Clock**
- **4,860 Kbits of Fast Block RAM**: This memory was used to store the image data both before and after processing.
- **16 Switches**: Used for user input to control filter selection.
- **16 LEDs**: Used to indicate the currently selected image filter.
- **12-bit VGA Output**: The board's VGA output was used to display the processed image in grayscale.

Design Considerations

Due to the constraints of the VGA's 12-bit (4 bits per color channel) color depth, the image pixel size was limited to 4 bits. Additionally, the capacity of the FPGA's block RAM was a limiting factor. To store both the original image and the processed image simultaneously, the image resolution was limited to 640 x 480 pixels.

Clock Configuration

Initially, the system was implemented using the 100 MHz clock provided by the FPGA. However, since the VGA output operates at 60 Hz, the 100 MHz clock was unnecessary and led to potential setup time violations. To address this, the design was re-implemented using a 10 MHz clock for the image processing block, and a slowed clock of 2.5 MHz for the VGA controller. This modification successfully resolved setup timing issues, while still maintaining proper system functionality and ensuring smooth image output.

Kernel Selection and Real-Time Filtering

Multiple 3x3 filter kernels were preloaded into the FPGA's ROM. The on-board switches were used to select the active filter, with the on-board LEDs indicating the currently selected filter. When a switch was toggled to change the filter, the FPGA applied the selected filter to the image in real time. The processed image was displayed on a connected monitor through the VGA interface as a 640 x 480 resolution 4-bit grayscale image.

This real-time filtering capability showcases the FPGA's ability to perform image processing with minimal latency, leveraging the parallel processing capabilities of the hardware to achieve high performance.

Hardware description - Link to the github repo and brief overview of code organization

Github Repository

https://github.com/nimsara1999/image_processing_accelerator_vhdl.git

Code Organization

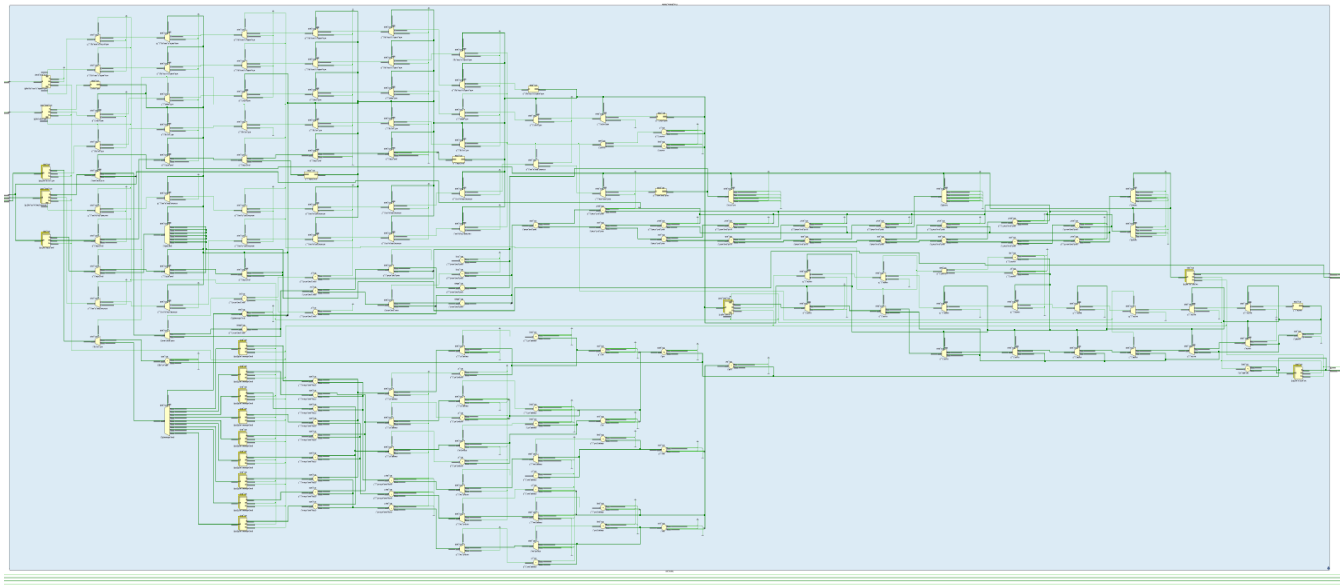
The project's hardware description is structured to maintain modularity, reusability, and ease of integration. The key components of the project include VHDL source files, an IP-defined True Dual Port Block RAM, and a Python script for image preprocessing.

1. **VHDL Source Files**:
    - `vga_controller.vhd`: This file implements the VGA controller responsible for generating synchronization signals (horizontal and vertical sync) and fetching pixel data from the Block RAM to output the image to a connected VGA display.
    - `pixel_window_fetcher.vhd`: This module extracts a 3x3 pixel window from the Block RAM, centered around a given pixel. It facilitates image processing operations such as applying filter kernels for edge detection and other effects.
    - `top.vhd`: This is the top-level design file that integrates the `vga_controller.vhd` and `pixel_window_fetcher.vhd` modules. It handles signal connections and coordinates data flow between these components.
2. **Block RAM (True Dual Port RAM)**:
    - The Block RAM is configured as a True Dual Port RAM using FPGA IP. The following configuration was used:
        - **Port A** (Image Processor Side):
            - **Read/Write Width**: 4 bits
            - **Read/Write Depth**: 700,000
            - **Write Enable Pin**: Mapped to a signal for dynamic control
        - **Port B** (VGA Side):
            - **Read/Write Width**: 4 bits
            - **Write Enable Pin**: Hardwired to 0, as it is used for read-only purposes
3. **Image Preprocessing (Python Script)**:
    - A Python script was developed to convert source images into a 640x480, 4-bit Black and White image. The converted image is stored in a `.coe` (Coefficient) file.
    - The `.coe` file serves as an initialization file for the Block RAM, allowing the FPGA to preload image data at the start of operation.

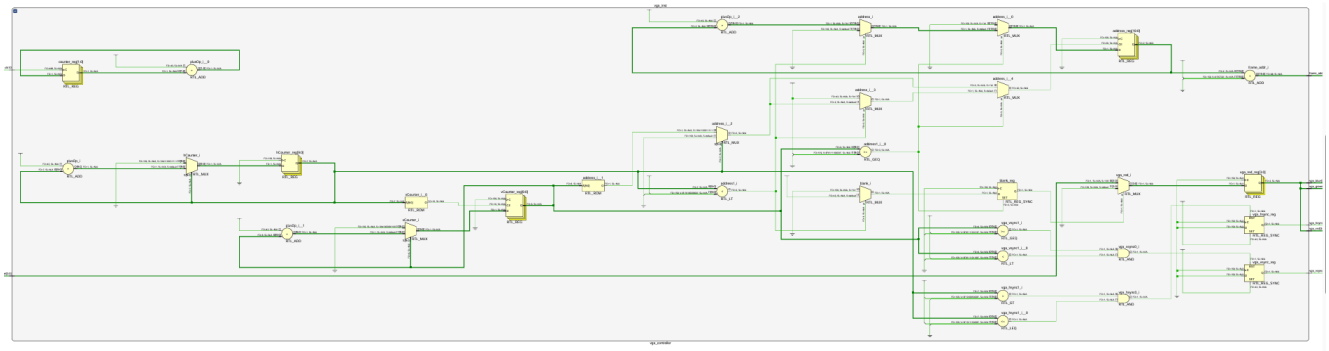
This hardware organization facilitates clear separation of concerns between image processing, data storage, and display. The modular design allows for easy future enhancements, such as the addition of new image processing modules or filter designs.

Module-vise RTL schematics

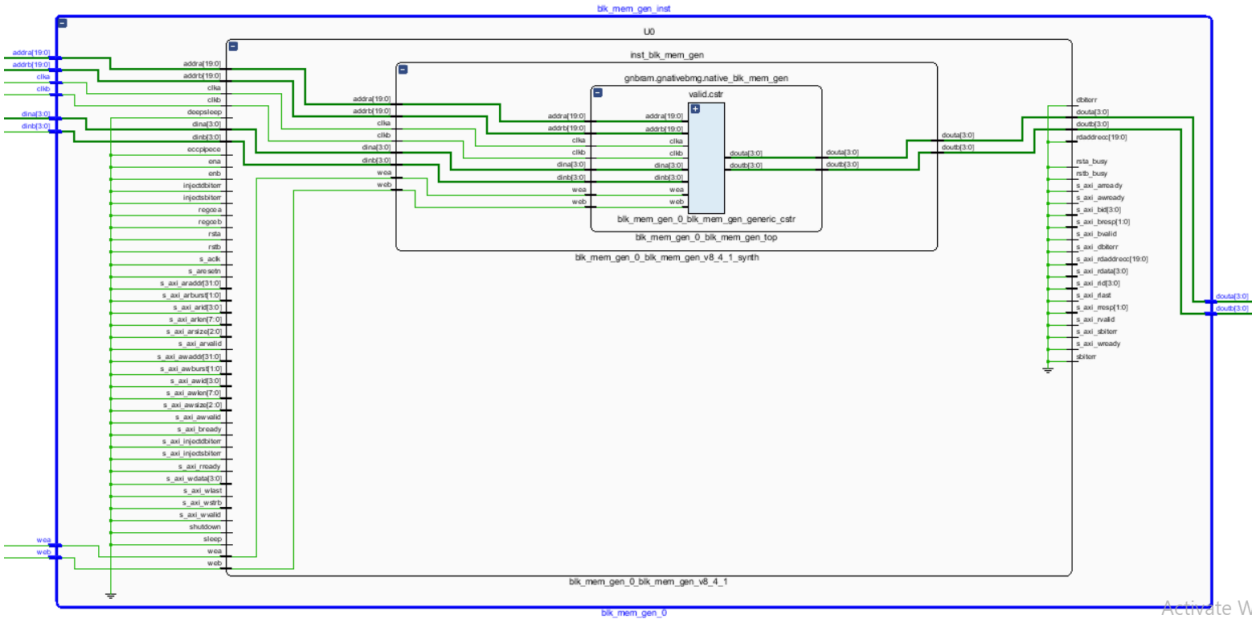
pixel_window_fetcher



vga_controller



Block memory



Constraints and Timing Analysis

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	22.002 ns	Worst Hold Slack (WHS):	0.098 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2100	Total Number of Endpoints:	2100	Total Number of Endpoints:	323

All user specified timing constraints are met.

Discussion

This project provided valuable practical experience in FPGA-based hardware design and implementation. Throughout the process, we gained in-depth knowledge of key concepts related to FPGA development, including:

- **Defining and Utilizing Block RAM**: We learned how to efficiently define and use block RAM for image storage and processing.
- **Crossing Clock Domains**: Handling multiple clock domains required understanding synchronization techniques to avoid timing issues.
- **Efficient RAM Usage**: We optimized RAM utilization by efficiently managing storage for both the input and processed images.
- **Multi-Port RAM Access**: We mastered the process of accessing RAM through multiple ports while managing Read-After-Write (RAW) and Write-After-Read (WAR) hazards.
- **VGA Communication**: We gained a deep understanding of VGA port communication, including the timing requirements for horizontal and vertical synchronization.

Hardware Selection Challenges

Initially, we began development on a BASYS 3 FPGA board. However, it became apparent that the block RAM capacity of the BASYS 3 board was insufficient for our design. The system required storage for both the input image and the processed image at the target resolution, which exceeded the board's memory capacity. This realization underscored the critical importance of evaluating hardware requirements before selecting an FPGA board for a project.

To address this limitation, we transitioned to the NEXYS A7 FPGA board, which offers approximately four times the block RAM capacity of the BASYS 3 board. With this additional capacity, we were able to store both the input and processed images simultaneously, enabling successful implementation of the project. This experience highlighted the importance of careful hardware resource planning and selection to meet project needs effectively.

Pending Unresolved Issues and proposed resolutions

1. OV7670 Camera Integration

Issue: The original plan was to use an OV7670 camera to feed the image directly into the FPGA for real-time processing. However, issues arose with the camera's focus and color accuracy, resulting in distorted and unreliable image inputs.

Proposed Resolution: To address this, we replaced the live image feed with a pre-stored Black and White image. This image is converted into a .coe file using a Python script and then loaded into the FPGA's RAM. For future iterations, enhanced debugging of the OV7670 camera's configuration settings, such as focus adjustments and color correction, could resolve the issue. Alternatively, a more robust image sensor with better focus control and color accuracy could be considered.

2. Color Image Processing

Issue: The current implementation supports only Black and White image processing. To extend the system to support color images, three color channels (Red, Green, and Blue) must be processed simultaneously, requiring three times the block RAM capacity currently used for grayscale images. The NEXYS A7 board does not have sufficient block RAM to accommodate this requirement.

Proposed Resolution: To achieve color image processing, future work could focus on developing optimization algorithms to reduce memory usage. For example, efficient compression schemes or pixel subsampling methods could be explored to reduce the storage requirement. Alternatively, a more advanced FPGA with higher block RAM capacity could be used, or an external RAM module could be interfaced with the FPGA to provide the required storage space.

Future prospects

The future development of this project presents several opportunities for enhancement and expansion. Below are the key prospects that could significantly improve the system's functionality and capabilities:

1. **Color Image Processing**:
   The current implementation processes only Black and White images. By introducing a compression algorithm, it would be possible to optimize memory usage and enable the system to support color images. The use of compression could reduce the storage requirements for Red, Green, and Blue (RGB) color channels, thereby allowing the NEXYS A7's block RAM to accommodate the necessary data. This enhancement would broaden the applicability of the system to more advanced image processing tasks.
2. **Real-Time Image Input via OV7670 Camera**:
   While the current system uses preloaded images from a .coe file, future iterations could reintroduce the OV7670 camera module for real-time image input. By achieving correct synchronization between the FPGA and the camera module, live image feeds could be processed directly on the FPGA. This would enable dynamic, real-time image processing applications such as video processing and real-time visual effects.

These future prospects offer pathways for scaling up the system's complexity and utility. By enabling support for color images and real-time processing, the project could transition from a functional prototype to a fully-featured image processing accelerator with broader practical applications.

References

https://www.fpga4student.com/2018/08/basys-3-fpga-ov7670-camera.html
https://www.youtube.com/watch?v=fqUuvwl4QJA