SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I5P109

Volume 12 Issue 5, 108-117, May 2025

Review Article

A Fast FPGA-Based Implementation of Linear and Non-Linear Image Filters

Riddhesh Veling[1], Aditya Vishwakarma[2], Shreyash Tiwari[3], Ravindra Chaudhari[4]

[1,2,3,4]*Department of Electronics and Telecommunication, St. Francis Institute of Technology, Maharashtra, India.*

[1]*Corresponding Author : velingriddhesh@gmail.com*

Received: 06 March 2025 Revised: 08 April 2025 Accepted: 09 May 2025 Published: 27 May 2025

Abstract *- Real-time image processing plays a crucial role in various domains, including medical imaging, surveillance, and autonomous systems, where the demand for efficient hardware acceleration is paramount. Field-Programmable Gate Arrays (FPGAs) emerge as a viable solution owing to their ability to perform parallel processing and their low-latency characteristics. This study introduces an FPGA-based implementation of both linear and non-linear image filters that are specifically optimized for real-time applications. The methodology employs a coefficient file (.coe) generation technique to facilitate efficient sum-of-product calculations and swift pixel ordering within a 3×3 window, all executed within a single clock cycle. The design, implemented on a Basys-3 FPGA using Verilog HDL and synthesized in Xilinx Vivado, achieves a processing latency of 0.04 ms at a clock frequency of 464 MHz while deliberately avoiding the utilization of FPGA DSP blocks. The paper provides a comprehensive account of the methodology to ensure reproducibility, detailing preprocessing steps, data management, and the experimental framework. The results indicate that the proposed architecture enhances computational efficiency without compromising image quality, thereby making it highly suitable for real-time applications in FPGA-based image processing.*

Keywords *- Digital signal processing, Field-programmable gate arrays, Image enhancement, Linear filters, Non-linear filters.*

1. Introduction

Field-Programmable Gate Arrays (FPGAs) have become a key platform in response to the growing demand for real-time image processing solutions. This paper highlights the benefits of using FPGA technology for real-time image processing, focusing on its high processing speed, parallel processing capabilities, and low latency, especially within the Basys3 FPGA framework. Image enhancement is a crucial preprocessing operation in various image-processing fields like medical imaging, surveillance, remote sensing, etc.

In real-time process, these operations must be executed in less amount of time compared to other key algorithms executions time. The proposed liner and non-linear filter architectures are combined and designed using efficient parallel processing architecture with excellent memory management techniques. It helps to minimize the overall computational latency with good visual quality of the output image. Image filters are also used to remove various noises like Gaussian, Salt and Pepper, etc.

With its FPGA Basys3 framework, users can execute various image enhancement techniques which utilize convolution-based procedures through a toolbox-free process. The FPGA Block RAM system operates on binary images to generate filtered output following various functions such as color conversion, blur effects, sharpening, edge detection, brightness modifications, and artistic visualization based on user selection. The processed images appear in real time on a VGA display, which provides instant feedback to the user.

FPGA design optimization uses Verilog language to achieve efficient parallel operations and maximize memory performance. Python enables user-friendly digital image preprocessing, allowing FPGA processors to work with binary data. Developments of FPGAs and optimizations proceed through the Vivado software suite alongside the platform for debugging operations.

Researchers study FPGA-based real-time image enhancement to improve the performance capabilities of image enhancement systems that operate in multiple sectors. Real-time image processing technological advancements enable safer unmanned car travel, enhance crop farm efficiency, and create better VR and augmented reality settings.

2. Related Works

In recent years, there has been a significant surge in research dedicated to leveraging the capabilities of Field-Programmable Gate Arrays (FPGAs) for implementing image enhancement techniques. To demonstrate the potential of

hardware acceleration in real-time image processing applications, Kumar et al. [1] used FPGA platforms to implement various image enhancement techniques. Complementing this, Nirmala et al. [2] conducted a comprehensive review focusing on FPGA-based image enhancement techniques, providing a nuanced understanding of methodologies and challenges. Their study served as a valuable resource for researchers and practitioners seeking insights into FPGA-based image enhancement strategies.

Moreover, Patel et al. [3] significantly contributed by exploring the utilization of Verilog HDL for FPGA implementation, offering a robust framework for enhancing image quality while capitalizing on hardware acceleration. This approach not only demonstrated promising results but also underscored the versatility of FPGA platforms in addressing image enhancement challenges.

Expanding on this foundational research, recent studies have delved deeper into specific applications of FPGA-based image enhancement techniques. For instance, [4] focused on real-time FPGA implementation, emphasizing the simultaneous realization of multiple image enhancement techniques to bolster processing efficiency. Their work shed light on the importance of optimizing FPGA resources to accommodate the computational demands of concurrent image enhancement algorithms. Ramyashree et al. [5] contributed to the body of knowledge by presenting FPGA implementation using a System Generator for contrast stretching in image enhancement. Their work emphasized the utilization of FPGA technology to efficiently implement specific image enhancement algorithms, further expanding the scope of FPGA-based image processing applications.

Shandilya [6] proposed a tailored FPGA-based approach for automatic vehicle number plate detection, emphasizing the pivotal role of image enhancement in enabling robust detection systems. This targeted application highlighted the practical implications of FPGA-accelerated image enhancement in real-world scenarios. Additionally, Narula [7] demonstrated the versatility of FPGA platforms by presenting implementations using Verilog HDL, showcasing the adaptability of FPGA-based image enhancement techniques across various domains. Moreover, AlAli et al. [8] contributed to the literature by exploring FPGA-based implementation of image processing algorithms, emphasizing the broader applicability of FPGA technology in image processing tasks beyond enhancement.

Furthermore, the work by Sowmya [9] presented FPGA implementation of image enhancement algorithms, enriching the understanding of FPGA-based image processing methodologies and their practical implications. The shift from traditional software-based simulations to hardware-based implementations for image enhancement is emphasized in [10], which explores FPGA-based techniques using Verilog HDL. The study highlights the implementation of thresholding, contrast adjustment, brightness control, and inversion on a Spartan-6 FPGA, demonstrating significant improvements in image quality and processing speed. [11] discusses various methodologies for noise removal in image processing, highlighting the challenges of implementing effective filters in FPGA hardware. It explores traditional median filter variants and advanced non-linear filters, such as those designed via Cartesian Genetic Programming (CGP), which improve filtering quality and hardware efficiency. [12] addresses noise removal in retinal images using an enhanced median filter algorithm on FPGA, which is crucial for diagnosing conditions like diabetic retinopathy. It optimizes noise reduction and processing speed by incorporating diagonal, vertical, and horizontal elements in a 3x3 window, using a sorting network for superior noise suppression and edge preservation compared to standard filters.

These studies collectively underscore the growing significance of FPGA technology in advancing image enhancement capabilities and paving the way for innovative applications in image processing.

3. Image Refinement Techniques

The image enhancement techniques implemented on FPGA board can be classified into 6 categories:

3.1. Color Transformation

Fundamental operations altering the color composition of images for improved visual representation.

3.1.1. RGB to Gray

Converting a color image to grayscale involves splitting each 24-bit pixel into red, green, and blue components. By adjusting these channels using right shift operations to represent 28.1%, 56.2%, and 9.3% of their respective colors [13], the color intensities are redistributed to create a balanced grayscale representation. The adjusted values are then combined to create an 8-bit grayscale representation for each pixel, allowing the entire image to be converted to grayscale.

$$\boldsymbol{Gray_{op}} = (\boldsymbol{R} \gg \mathbf{2}) + (\boldsymbol{R} \gg \mathbf{5}) + (\boldsymbol{G} \gg \mathbf{1}) + (\boldsymbol{G} \gg \mathbf{4}) + (\boldsymbol{B} \gg \mathbf{4}) + (\boldsymbol{B} \gg \mathbf{5}) \quad (1)$$

3.1.2. Color Inversion

Color inversion in image processing involves transforming the RGB components of each pixel in a color image to their respective inverted values. This is achieved by subtracting the original intensity values of the red, green, and blue channels from 255. This operation reverses the color scheme of the image, making bright areas dark and vice versa. Applying this technique to all pixels ensures a uniform and complete inversion of colors, enhancing the overall image. In this context, R, G, and B represent the input RGB components, and R_{op}, G_{op}, B_{op} represent the output components.

$$R_{op} = 255 - R$$
$$G_{op} = 255 - G \quad (2)$$
$$B_{op} = 255 - B$$

Fig. 1 Color inversion filters output

3.2. Brightness Adjustment

Techniques for modifying the overall brightness level of images to enhance or diminish illumination.

3.2.1. Increase Brightness

The Increase Brightness technique employs a simple yet effective method to enhance the luminosity of an image uniformly. By adding a predetermined adjustment factor 'g' to the intensity of each pixel, the technique effectively brightens the entire image. The formula adopted from [9, 14] ensures that intensity values remain within the valid range of 0 to 255. Here, $I_k(r, c)$ and $J_k(r, c)$ are input and output pixel values.

$$J_k(r,c) = \begin{cases} I_k(r,c) + g & , if\ I_k(r,c) + g \le 255 \\ 255 & , if\ I_k(r,c) + g > 255 \end{cases} \quad (3)$$

3.2.2. Decrease Brightness

The Decrease Brightness technique operates by uniformly reducing the luminance of an image. This method diminishes the intensity of each pixel by a predetermined adjustment factor $'g'$. By systematically lowering the brightness across the image, the technique effectively attenuates the luminosity of brighter regions while preserving the visual details and contrast within the image.

$$J_k(r,c) = \begin{cases} I_k(r,c) - g & , if\ I_k(r,c) - g \ge 0 \\ 0 & , if\ I_k(r,c) - g < 0 \end{cases} \quad (4)$$

3.3. Edge Detection

Provide sharp intensity transitions for the purpose of object boundary and structural feature identification.

3.3.1. Outline (Laplacian Filter)

Edge detection utilizing the Laplacian Filter is pivotal in image processing, as it allows for the precise localization of edges and contours. By amplifying the intensity of the central pixel and subtracting the weighted sum of neighbouring pixel intensities, this method effectively highlights areas of rapid intensity changes [15]. The Laplacian Filter is particularly adept at detecting edges regardless of their orientation or thickness, making it a versatile tool for edge detection tasks in various image processing applications. However, it has certain drawbacks, including sensitivity to noise, the tendency to produce thick edges, and susceptibility to gradient reversal issues. The Laplacian Filter kernel can be seen in Figures 2(a) and (b).

3.3.2. Sobel Edge Detection

The Sobel Edge Detection algorithm is a fundamental technique in image processing aimed at highlighting edges within an image. This method involves the application of the Sobel operator in both the horizontal (x) shown in Figure 2(c) and vertical (y) shown in Figure 2(d) directions to detect changes in intensity indicative of edges [16, 17]. By convolving the image with separate Sobel kernels for x and y directions, the algorithm computes the gradient magnitude of each pixel, representing the rate of change of intensity. Subsequently, the gradient magnitudes' Root Mean Square (RMS) value in both directions is calculated to create the final edge-emphasized image. This approach effectively enhances edges while suppressing noise, enabling precise edge detection and boundary delineation in various image-processing applications.

0	-1	0
-1	4	-1
0	-1	0

(a)

-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

-1	-2	-1
0	0	0
1	2	1

(c)

-1	0	1
2	0	2
-1	0	1

(d)

Fig. 2 Edge Detection kernels (a) Laplacian kernel (used for horizontal and vertical edge differences), (b) Laplacian kernel (the one used here), (c) G_x horizontal component, and (d) G_y vertical component.

3.4. Blurring

Averaging the pixel values of neighboring pixels to reduce image detail and noise to improve the appearance.

3.4.1. Average Blurring

Average blurring is a widely used image filtering technique employed to reduce noise and smooth out images. This method involves convolving the image with a kernel where each element represents a weighted average of its neighbouring pixels. Typically, the kernel, as shown in Figure 3. (a), is a square matrix with equal weights assigned to each element, resulting in a uniform blur effect across the image.

Average blurring is effective in reducing high-frequency noise while preserving overall image structure.

3.4.2. Motion Blurring(xy)

Motion blurring, often employed in image processing and computer graphics, simulates an object's motion effect by averaging the pixel values along a specific direction. As shown in Figure 3(b), a convolution kernel employs diagonal square matrix organization with nonzero elements because the x and y dimensions blend together in the situation (xy).

The weight associated with a pixel in the image, e.g. the element of the kernel, is high when the pixel is closer to the movement direction. Consequently, motion blurring in the xy direction leads to streak-like artifacts in the image, which mimic the effect of objects moving in horizontal and vertical directions.

3.4.3. Motion Blurring (x)

Motion blurring in the x direction is a very common method in image processing to simulate object movement in the horizontal direction. Convolution with a kernel that averages pixel values along the x-axis results in streaks produced in the direction of motion, thus making up this method. In Figure 3(c), the kernel contains a row having non-zero elements, each of which represents the weight associated with the neighbouring pixels. It blurs the pixels with greater weights assigned closer to the motion direction.

3.4.4. Weighted Average Blurring

Weighted Average Blurring is one of the most used image processing tools that reduces noise and smooths the images while not distorting the important features. This method first involves convolving the image with a bell shape-weighted average kernel, having pixel weight given by a bell-shaped curve [18].

In Figure 3(d), the kernel's central pixel has the highest weight, and its value gradually falls as it gets away from the center. The structure is, therefore, low frequency noise reducing while providing an essential structural detail while causing that gentle, isotropic blurring effect. Unlike uniform averaging, which weights the contribution of all the neighboring pixels equally, weighted averaging gives more weight to the neighboring pixels closer to the averaged pixels to avoid excessive detail loss.

This is an often-used approach in medical imaging, object detection and photography enhancement to reduce noise without causing image sharpness to decline significantly. This technique is popular because, typically, Gaussian kernels are used because they are efficient in terms of balancing out noise reduction with feature preservation; thus, it can be used as a pre-processing technique in computer vision tasks such as feature extraction and edge detection.

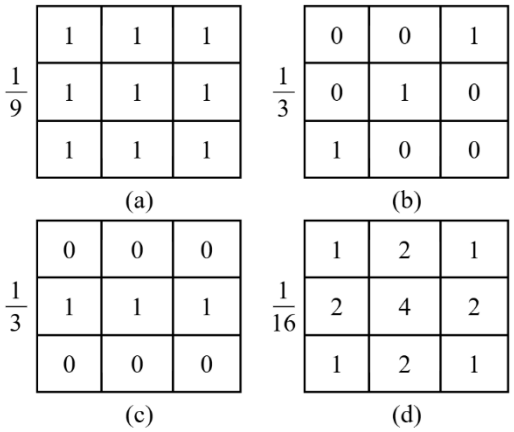

Fig. 3 Blurring Kernels (a) Average blurring kernel, (b) Motion blurring xy kernel, (c) Motion blurring x kernel, and (d) Weighted average blurring kernel.

3.5. Artistic Filters

Artistic effects were put onto the images to make them look different and unique, giving them an artistic style.

3.5.1. Embossing

Image enhancement by embossing is a particular technique in which edges in an image are emphasized through embossing. The embossing technique simply convolves the picture using an embossed kernel, typically a small square matrix. Figure 4(a) shows a kernel that organizationally features a symmetrically negative and positive parts, related to the zero parts, which is in turn surrounded by a negative core. When applied to the picture, this kernel contrasts the pixel brightness of adjacent regions for the sake of highlighting edges.

3.5.2. High Boost Filtering (Sharpening)

Image enhancement with sharpening is an important technique for making images clearer and more detailed by enhancing the edge contrast. Here we convolve the image with a sharpening kernel, making the image appear sharper by enhancing the difference in pixel intensities between neighboring regions of the image. Figure 4(b) shows that the sharpening kernel is a positive-valued central element surrounded by negative and zero-valued elements in a symmetric configuration. Thus, when we apply it to the image, the sharpening kernel increases the intensity of edges so that they appear sharper and more distinct.

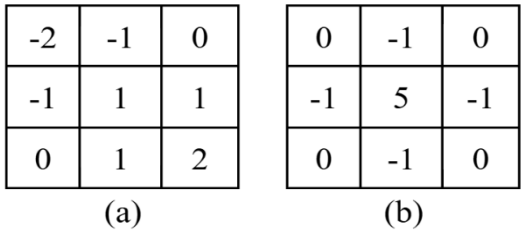

Fig. 4 Artistic Kernels (a) Emboss filter mask, and (b) High boost filter mask.

3.6. Non-Linear Filters

Modification of pixel values using complex neighbours relations preserves the edges while removing noise.

3.6.1. Median Filter

The median filter kernel slides a window over the signal or image and replaces each pixel's value with the median of the pixels in the window. In the first step, you need to define the size of the kernel, for example, 3x3 or 5x5. While the kernel is being traversed over the image pixel by pixel, all the pixel values in the window are being stored. The list of these values is then sorted using the merge sorting technique, which divides the list into smaller sublists, sorts them, and combines them back with another in ordered form. After sorting, the original pixel value is replaced with the median value as in this sorted list, or (if the list has an even number of elements) the average of the middle two values. This method provides a good result in image smoothing and noise reduction.

4. Proposed Design Flow

As opposed to the image input technique as in [1, 9, 19], the Proposed Design Flow to convert an input image to some intermediate format for the processing is divided into 2 stages respectively:

4.1. Initialization

The initialization stage consists of two Python files utilized for converting the input image into a format compatible with the Basys-3 FPGA board, as shown in Figure 6. The first Python script is responsible for splitting the given input image into nine smaller overlapping sections, ensuring that the FPGA can process localized pixel information efficiently. This division allows for parallel processing of different sections of the image, optimizing computational performance. Additionally, the script applies necessary padding to the edges of the image to maintain uniform dimensions, preventing data loss at the borders. Each of these segmented images is stored temporarily in an array before being passed to the second Python script for further processing.

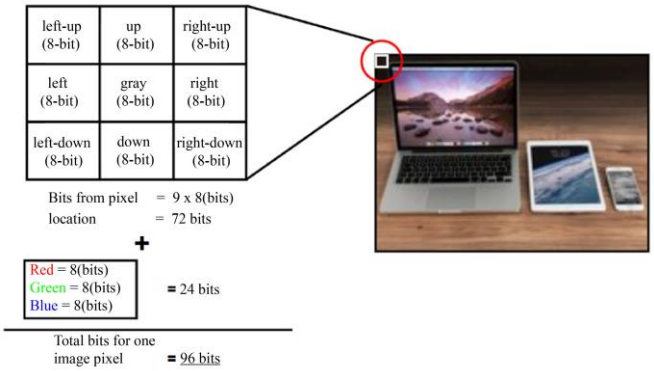

Fig. 5 Bits formation from one pixel of the input image

The second Python script is responsible for generating and initializing the coefficient (.coe) file, which is essential for storing image data in FPGA memory. This script defines the necessary preamble for setting the memory initialization radix and memory initialization vector. It goes over each pixel of the grey scale image and its shifted versions, iterate over their intensity value, and converts to binary string. They are concatenated to retain structured memory access, formatted in a .coe file structure, and stored sequentially. The binary data resulting after processing each row of pixels is appended to .coe file for the proper organization for FPGA-based filtering operations. Also, the RGB value of the original .png image is extracted and saved at the end of each row to keep the color. The file is then finalized and stored in the correct directory where the file can be integrated into the FPGA memory. First, it guarantees that the FPGA obtains pre-processed image data directly, so that image enhancement algorithms can be run in real time.

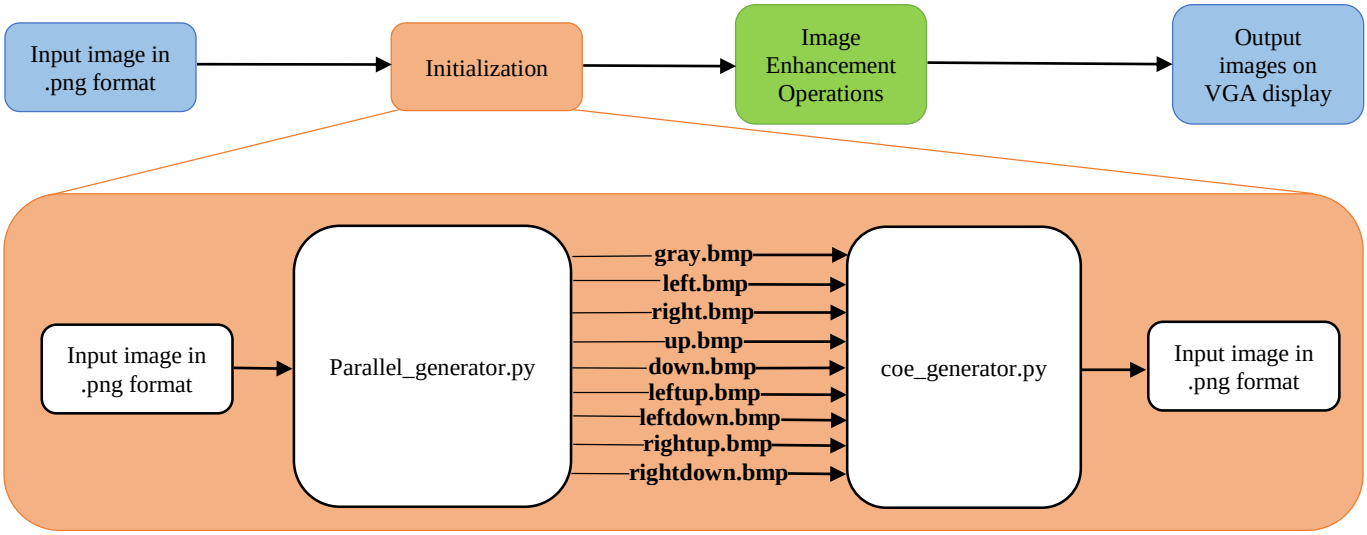

Fig. 6 Flow diagram for Initialization Stage

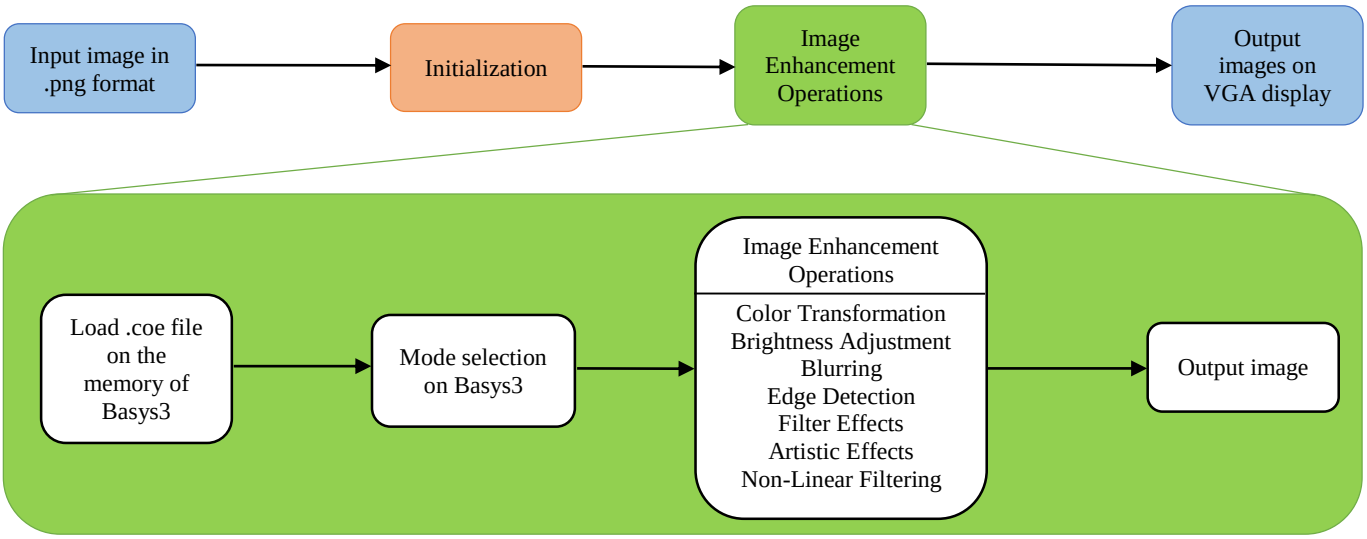

Fig. 7 Flow diagram for image enhancement operations stage

4.2. Image Enhancement Operations

The .coe file generated in the initialization stage is utilized by the Verilog code, as shown in Figure 7. For example, if the image has a size of 160x115, then the number of rows in the .coe file will be equal to the size of the image (it will be 18400 rows). The memory is loaded with the .coe file. At each positive clock edge, one row of this file is stored in a temporary array. Depending on what operation is selected, switches on Basys3 are used to access elements of this array. The board does various image enhancement operations according to the mode chosen. The applied image enhancements result in a processed image rendered on a VGA display connected to the Basys3, allowing real-time visualization of the effects of the applied image enhancements.

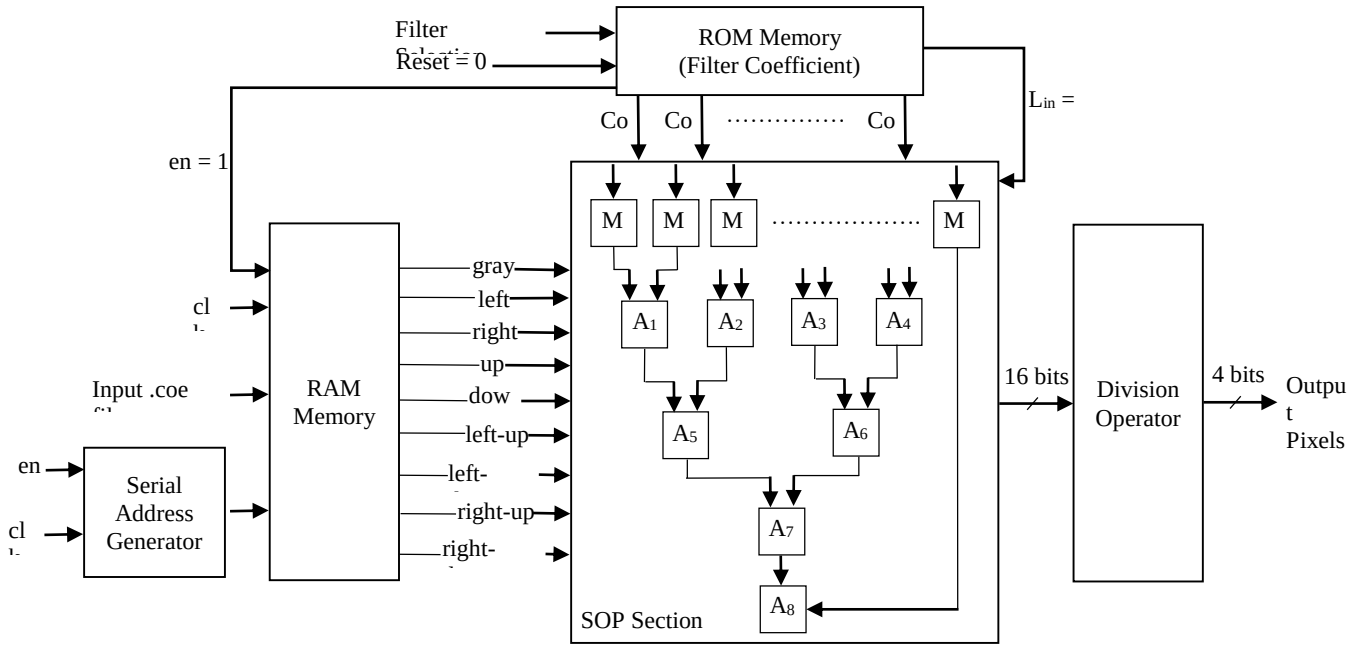

Fig. 8 Architecture of proposed design

Initially, the entire .coe file is stored in a RAM memory cell, as seen in Figure 8. At each row memory location, all 9-pixel values are stored with the variable ‘gray’ as the centre for every 3x3 selection, and the surrounding 8-pixel values are also stored in the same row. Similarly, all different types of filter coefficients are stored in a ROM memory cell. According to the 4-bit filter selection input, a particular type of filter is selected. When reset = 0, all 9 coefficients are assigned parallel to the variables Co_1, Co_2, ..., Co_9. Whenever a linear filter is selected through the filter select input, the

variable Lin is set to 1, and for a non-linear filter, Lin is set to 0. Additionally, another variable, en, is set to 1 for both types of filters.

When en = 1, at the next positive clock edge, the input address generator generates the first address location. Subsequently, the next address location is sequentially generated at every positive clock signal. In this way, at each clock pulse, all 9 values are assigned in parallel to the variables of the SOP section.

The Sum of Products (SOP) of 9 coefficients and 9 pixels is performed using 9 multipliers and 8 adder units. In the end, the final output value is truncated to a specific number of bits using a division operation facilitated by a right shift operation. This output is then sequentially applied to a display device through the FPGA board.

When L_{in} = 0, meaning that non-linear filtering using the median filter is selected, the separated input pixels are taken as input to the merge sort algorithm. The merge sort algorithm works on the principle of divide-and-conquer technique. The pixels are already divided or separated and assigned to the given 9 variables, as shown in Figure 9; now, these are sorted to find the median value at the 5th position.

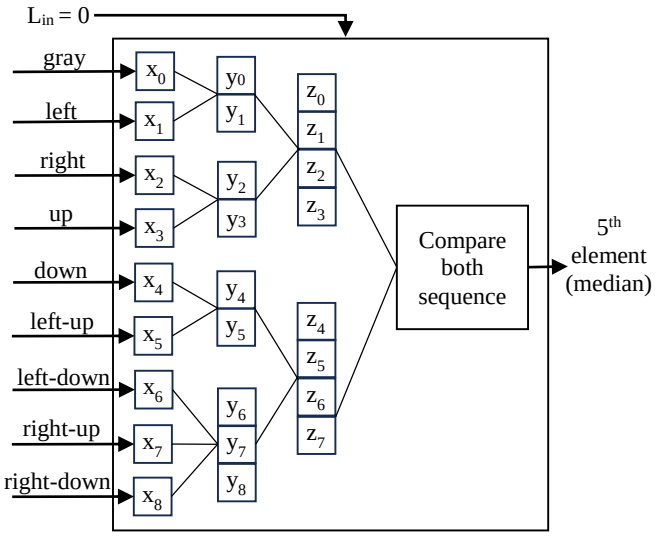

Fig. 9 Non-linear median filtering using merge-sort algorithm

Three vectors are assigned, namely x, y, and z, with sizes 9 and each 8-bit length. Only their indexes are controlled to sort the sequences within 3 stages. In the first stage, 2 samples are compared to find the minimum and maximum value, and then it is arranged in vector 'y' as shown in Figure 9.

Similarly, in the 2nd stage, 4 samples are compared, and their results are stored in vector 'z'. In the last stage, 2 sequences with lengths 4 and 5 samples, respectively, are compared and determine the 5th smallest value, which will be the median value of the input sequence.

5. Simulation Results

The Artix-7 Basys3 FPGA Board was used for synthesis purposes. Within the Basys3 FPGA framework, users can choose from various image enhancement operations using physical switches/buttons, including filters, color transformations, and blurring. The entire code is written in Verilog HDL language, and simulation/synthesis is done with Vivado ML Edition 2022.2 version. The 4 switches represent 4 bits for each image enhancement mode. Toggling switches/buttons enables smooth transitions between modes, facilitating customization of image processing workflows, as demonstrated in the presented outcomes.

Image processing techniques encompass various methods to enhance and transform images. Figure 10 shows grayscale conversion adjusts the red, green, and blue channels to create an 8-bit representation. Color inversion subtracts RGB values from 255, while brightness adjustments add or subtract a predetermined factor to each pixel's intensity.

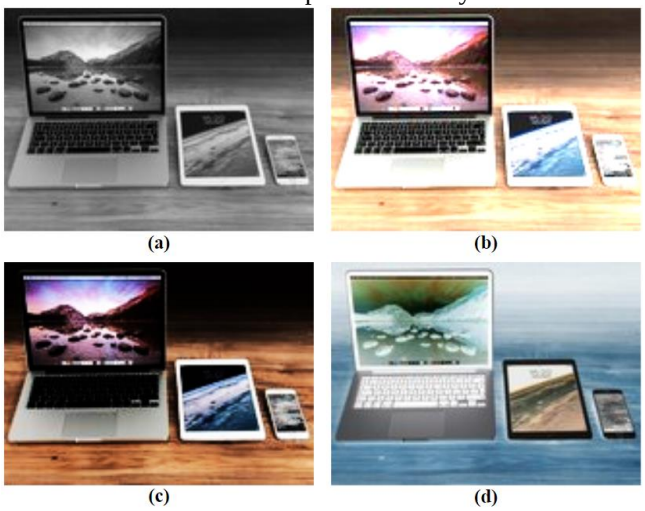

Fig. 10 (a) 0000 for RGB to gray, (b) 0001 for increase brightness, (c) 0010 for decrease brightness, and (d) 0011 for color inversion.

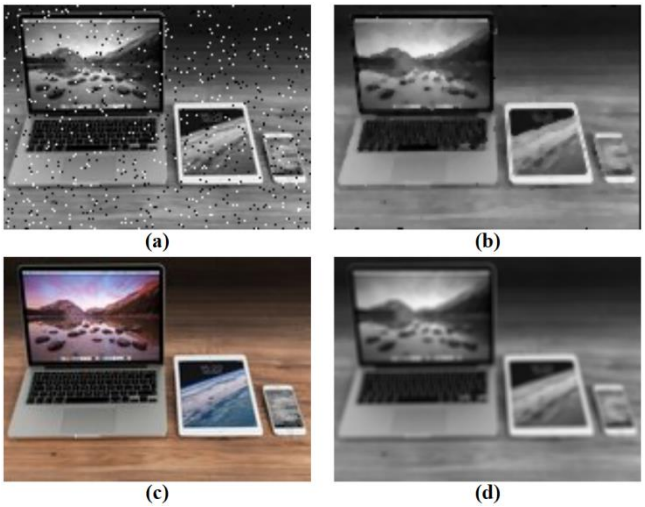

Fig. 11 (a) Salt and pepper noise image, (b) 0100 for median filtered image, (c) 0111 for Original image, and (d) 1000 for average blurring.

Figure 11 shows Salt and pepper noise is introduced to the image, and median filtering replaces each pixel with the median of its neighbors, both utilizing the FPGA's parallelism. Average blurring replaces each pixel with the average of its neighbors through parallel convolution, similarly employed in Sobel and Laplacian edge detection for highlighting edges.

Figure 12 shows motion blurring in both (x & y) and (x) directions involves convolving the image with a motion-simulating kernel, while embossing uses a convolution kernel to emphasize intensity differences, creating a 3D effect.

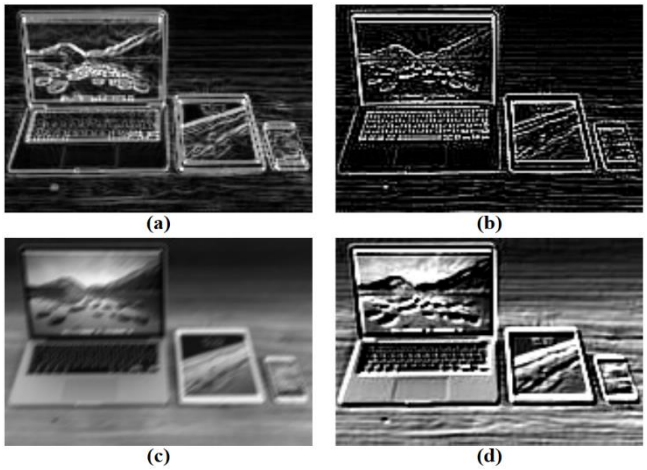

Fig. 12 (a) 1001 for sobel edge detection, (b) 1010 for outline (laplacian edge detection), (c) 1011 for motion blurring (xy), and (d) 1100 for embossing.

As shown in Figure 13, high boost filtering enhances edges by amplifying high-frequency components, requiring parallel convolution and arithmetic operations. Gaussian noise introduces normally distributed perturbations, and Gaussian blurring applies a weighted average, both efficiently handled by FPGAs using their parallel processing strengths.

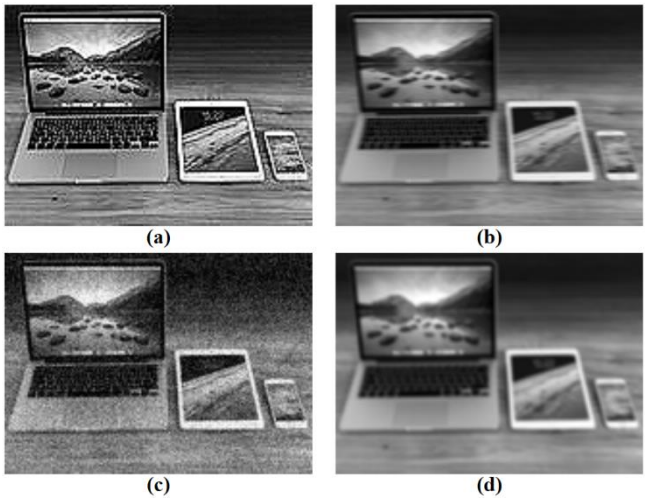

Fig. 13 (a) 1101 for sharpening (high boost filtering), (b) 1110 for motion blurring (x), (c) Gaussian noise image, and (d) 1111 for weighted average blurring (gaussian blurring)

The maximum clock frequency used is 464 MHz. The propagation delay in getting one output pixel after completing the entire convolution operation is 2.155 nsec. So, the latency (number of clock cycles) for the entire image is equal to the total no. of pixels of an image. In this case, the latency delay is 0.04 msec.

Table 1. Hardware resource utilization summary

Resource	Utilization	Available	Utilization %
LUT	838	20800	4.03
FF	61	41600	0.15
BRAM	49	50	98.00
IO	20	106	18.87
DSPs	0	90	0

The resource utilization of the FPGA implementation is summarized in Table 1, which presents an overview of the hardware resource consumption. The table highlights the utilization of key FPGA components, including Look-Up Tables (LUTs), Flip-Flops (FFs), Block RAMs (BRAMs), Input/Output (IO) blocks, and Digital Signal Processing (DSP) slices. The percentage utilization of each resource is also provided to illustrate the efficiency of hardware usage. Notably, the BRAM utilization is significantly high, nearing 98%, indicating that memory resources are a primary design component. On the other hand, the DSP utilization remains at 0%, suggesting that the implemented filtering techniques primarily rely on logical operations rather than DSP-based computations.

Table 2. Slice logic utilization summary

Resource	Utilization	Available	Utilization %
SLICE LUTs	194	20800	0.93
LUT as Logic	194	20800	0.93
LUT as Memory	0	9600	0
Slice Registers	8	41600	0.02
Register as FF	8	41600	0.02
Register as Latch	0	41600	0
F7 Muxes	90	16300	0.55
F8 Muxes	0	8150	0

Further insights into the logical utilization of FPGA slices are presented in Table 2. This table details the usage of slice LUTs, registers, multiplexers (Muxes), and different configurations of logic resources. The LUTs are primarily used as logic elements, with no utilization as memory blocks. Slice registers, which play a crucial role in sequential logic

design, show minimal usage, indicating that the design relies more on combinational logic. The utilization of multiplexers, particularly F7 and F8 Muxes, demonstrates how efficiently routing and selection operations are managed within the FPGA fabric. These tables collectively provide a comprehensive view of how efficiently the FPGA resources are allocated and utilized to implement the proposed image filtering system.

6. Conclusion

The implementation of FPGA for image enhancement is a Verilog-based project designed for the Basys3 FPGA, which is aimed at facilitating image-enhancing operations such as convolution on input images. It leverages Block RAM to store images and displays them through a VGA interface. Development is carried out using Verilog and Python with the Vivado software suite. The project offers two primary implementations: one for the transfer of images between a PC and the FPGA and another for the display of processed images on a monitor. The VGA interface is specifically configured for a 480p display with a 60Hz refresh rate. To initialize the block RAM in Xilinx FPGA designs, a .coe file is provided. The project's development structure supports basic image enhancement and convolution-based operations, each requiring specific implementation approaches. Python scripts are utilized to generate .coe files tailored to these operations. Additionally, incorporating hardware accelerators or co-processors could further enhance the system's performance and enable it to tackle more computationally intensive tasks.

References

[1] Karan Kumar, Aditya Jain, and Atul Kumar Srivastava, "FPGA Implementation of Image Enhancement Techniques," *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2009*, Wilga, Poland, vol. 7502, pp. 1-10, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[2] S.O. Nirmala, T.D. Dongale, and R.K. Kamat, "Review on Image Enhancement Techniques: FPGA Implementation Perspective," *International Journal of Electronics Communication and Computer Technology*, vol. 2, no. 6, pp. 1-9, 2012. [Google Scholar] [Publisher Link]

[3] Sagar Patel et al., "Image Enhancement on FPGA Using Verilog," *International Journal of Technical Innovation in Modern Engineering & Science*, vol. 5, no. 3, pp. 2455-2585, 2019. [Google Scholar] [Publisher Link]

[4] Muhammed Yildirim, and Ahmet Çinar, "Simultaneously Realization of Image Enhancement Techniques on Real-Time FPGA," *2019 International Artificial Intelligence and Data Processing Symposium*, Malatya, Turkey, pp. 1-6, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] B.H. Ramyashree, R. Vidhya, and D.K. Manu, "FPGA Implementation of Contrast Stretching for Image Enhancement Using System Generator," *2015 Annual IEEE India Conference*, New Delhi, India, pp. 1-6, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[6] Rahul Shandilya, and R.K. Sharma, "FPGA Implementation of Image Enhancement Technique for Automatic Vehicles Number Plate Detection," *2017 International Conference on Trends in Electronics and Informatics*, Tirunelveli, India, pp. 1-5, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[7] Mandeep Singh Narula, and Nishant Singla, "FPGA Implementation of Image Enhancement Using Verilog HDL," *International Research Journal of Engineering and Technology*, vol. 5, no. 5, pp. 1794-1797, 2018. [Google Scholar] [Publisher Link]

[8] Mohammad I. AlAli, Khaldoon M. Mhaidat, and Inad A. Aljarrah, "Implementing Image Processing Algorithms in FPGA Hardware," *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies*, Amman, Jordan, pp. 1-5, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[9] S. Sowmya, and Roy Paily, "FPGA Implementation of Image Enhancement Algorithms," *2011 International Conference on Communications and Signal Processing*, Kerala, India, pp. 584-588, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[10] Narayan A. Badiger et al., "FPGA Implementation of Image Enhancement Using Verilog HDL," *International Research Journal of Engineering and Technology*, vol. 7, no. 6, pp. 5663- 5668, 2020. [Google Scholar] [Publisher Link]

[11] Zdenek Vasicek, Michal Bidlo, and Lukas Sekanina, "Evolution of Efficient Real-Time Non-Linear Image Filters for FPGAs," *Soft Computing*, vol. 17, pp. 2163-2180, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[12] C.K. Priyanka, "Median Filter Algorithm Implementation on FPGA for Restoration of Retina Images," *International Journal of Innovative Science, Engineering & Technology*, vol. 3, no. 5, pp. 415-420, 2016. [Google Scholar] [Publisher Link]

[13] Kaushal Kumar, Ritesh Kumar Mishra, and Durgesh Nandan, "Efficient Hardware of RGB to Gray Conversion Realized on FPGA and ASIC," *Procedia Computer Science*, vol. 171, pp. 2008-2015, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[14] R. Dhanabal et al., FPGA Based Image Processing Unit," *2015 IEEE 9th International Conference on Intelligent Systems and Control*, Coimbatore, India, pp. 1-4, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[15] Ghassan Mahmoud Husien Amer, and Ahmed Mohamed Abushaala, "Edge Detection Methods," *2015 2nd World Symposium on Web Applications and Networking*, Sousse, Tunisia, pp. 1-7, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[16] Girish Chaple, and R.D. Daruwala, "Design of Sobel Operator Based Image Edge Detection Algorithm on FPGA," *2014 International Conference on Communication and Signal Processing*, Melmaruvathur, India, pp. 788-792, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[17] Zhang Jin-Yu, Chen Yan, and Huang Xian-Xiang, "Edge Detection of Images Based on Improved Sobel Operator and Genetic Algorithms," *2009 International Conference on Image Analysis and Signal Processing*, Linhai, China, pp. 31-35, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[18] James Coady et al., "An Overview of Popular Digital Image Processing Filtering Operations," *2019 13th International Conference on Sensing Technology*, Sydney, NSW, Australia, pp. 1-5, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[19] Tarek M. Bittibssi et al., "Image Enhancement Algorithms Using FPGA," *International Journal of Computer Science and Communication Networks*, vol. 2, no. 4, pp. 536-542, 2012. [Google Scholar]