Design and Implementation of Xilinx SoC FPGA Based Custom IP for Scharr Edge Detection in Real-Time Image Processing

1st Hema Chitra S
Associate Professor
Department of Electronics and Communication Engineering
PSG College of Technology
Coimbatore, India
shc.ece@psgtech.ac.in

2rd Jenet S
Student, ME (VLSI DESIGN)
Department of Electronics and Communication Engineering
PSG College of Technology
Coimbatore, India
23mv06@psgtech.ac.in

***Abstract*—Real-time image processing plays a crucial role in applications such as autonomous navigation, medical imaging, and industrial automation, where high-speed and accurate data analysis is essential. Traditional image processing approaches often rely on standard IP cores that process images in HEX format, limiting flexibility and requiring format conversions. In contrast, this work implements FPGA-based image processing operations with BMP files directly used as both input and output. The implemented system performs essential operations, including blurring and edge detection using Prewitt, Scharr, Sobel, and Laplacian methods. To evaluate the performance of these edge detection techniques, edge density analysis is performed, revealing that Scharr edge detection provides the most accurate results. Based on this analysis, a dedicated custom IP is developed for Scharr edge detection to enhance accuracy and efficiency. Furthermore, the power, area, and timing efficiency of the proposed system are examined, ensuring optimized resource utilization for real-time FPGA-based image processing applications. The modular and reusable IP cores offer flexibility for various real-time image processing tasks.**

Keywords*—*Custom IP, edge detection, real-time image processing, efficiency, image bluring, SoC FPGA.

I. INTRODUCTION

Image processing is a critical component in a wide range of modern technologies, enabling machines to interpret, analyze, and manipulate visual data. The ability to process and analyze images in real-time is particularly essential in fields such as autonomous vehicles, medical imaging, surveillance, industrial automation, and more. These applications require fast, efficient, and precise processing of visual data to make informed decisions and improve operational efficiency. Image processing operations include tasks such as noise reduction, edge detection, feature extraction, image enhancement, and object recognition, all of which are fundamental for extracting valuable information from raw visual data[1]. Traditional image processing systems rely on software-based solutions, which offer flexibility but often fail to meet the real-time performance and power efficiency requirements of many modern applications. These solutions can be computationally expensive, leading to delays that are unacceptable in time-sensitive applications[1]. Additionally, software implementations tend to consume significant computational resources, making them inefficient for applications that require continuous, high-speed data processing. In contrast, Field-Programmable Gate Arrays (FPGAs) provide a powerful alternative for implementing real-time image processing systems. FPGAs offer the ability to process data in parallel, significantly accelerating processing speeds while maintaining low power consumption[2]. These hardware platforms are highly configurable, making them ideal for developing custom image processing solutions that can be tailored to meet the specific needs of an application. FPGAs can implement highly optimized image processing operations by taking advantage of hardware parallelism, pipelining, and customized data flows, enabling faster and more efficient processing compared to traditional general-purpose processors.

One of the key advantages of FPGAs for image processing is their ability to implement custom IP (Intellectual Property) cores. Custom IP cores are specialized logic designs that can be implemented on FPGA hardware to perform specific tasks at high speeds with minimal resource consumption[2]. These cores can be reused and reconfigured for different image processing tasks, offering flexibility, scalability, and optimization in real-time systems. By leveraging FPGAs and custom IP cores, it becomes possible to achieve high-performance image processing solutions that can operate within the strict timing and power constraints required by real-time applications[4].

However, challenges remain in optimizing image processing workflows on FPGAs. These include minimizing resource usage, ensuring high throughput, and managing data efficiently, especially when handling large images or multiple image processing tasks simultaneously. The design of custom IP cores tailored to specific image processing operations, such as brightness adjustment, edge detection, and noise reduction, is essential to achieving the necessary performance and power efficiency in FPGA-based systems[3].

In this context, the use of FPGAs for image processing has become increasingly important, and the development of efficient, adaptable, and high-performance custom IP cores is a key area of research. Such solutions are critical for advancing real-time image processing in fields such as autonomous navigation, medical diagnostics, and industrial automation, where fast and accurate data analysis is essential for successful operation.

II. RELATED WORK

In [21], the design and implementation of an FPGA-based image processing system using the Zynq SoC were explored. The study focused on performing essential image processing operations, including brightness adjustment, inversion, and thresholding, utilizing Xilinx Vivado and SDK tools. The prototype was verified by interfacing the Zynq processing system with an OLEDrgb peripheral module through AXI interconnects. The system efficiently processed images in HEX format, converting BMP files to HEX before processing and reconverting the output. While the work demonstrated the feasibility of FPGA-based image processing, it primarily focused on fundamental image enhancement techniques.

FPGA-based image processing has seen significant advancements, with researchers exploring various techniques to improve speed, efficiency, and accuracy for real-time applications[2]. Edge detection, brightness adjustment, thresholding, and blurring are fundamental operations that have been widely studied and implemented. Among edge detection techniques, Sobel and Prewitt remain popular for their simplicity and computational efficiency, though they are limited in detecting diagonal edges and handling noise[6]. Scharr edge detection improves upon these by offering better rotational symmetry, making it more effective for detecting edges in complex or noisy images. Laplacian methods, which calculate second-order derivatives, provide higher sensitivity to intensity variations but are prone to noise amplification[8].

In [1], the authors developed an FPGA-based implementation of edge detection and image filtering using the Sobel and Canny algorithms. The study highlighted the trade-offs between computational complexity and edge detection accuracy, with Canny providing superior results at the cost of increased resource usage. This work demonstrated the importance of balancing algorithm complexity with hardware constraints.

In [2], the design of custom FPGA cores for brightness and contrast adjustments was explored, emphasizing the importance of resource optimization for real-time performance. The authors noted that traditional IP cores often suffer from inefficiencies when integrated into larger systems, necessitating tailored solutions for specific applications.

Recent works have also focused on integrating hardware accelerators with embedded processors for enhanced performance. For example, [3] presented a hardware/software co-design approach for image processing on the Xilinx Zynq platform. The system used the ARM processor for high-level control and FPGAs for parallel execution of tasks like grayscale conversion, edge detection, and filtering. This hybrid approach showed significant improvements in speed compared to pure software implementations.

In addition to hardware-based solutions, research has explored hybrid systems combining FPGAs with GPUs or CPUs for specific tasks. In [7], a hybrid system was developed to process high-resolution images using FPGAs for preprocessing (e.g., filtering and edge detection) and GPUs for computationally intensive tasks like object recognition. This division of labor leveraged the strengths of each platform to achieve a balance between speed and resource utilization.

Despite these advancements, challenges remain in optimizing power consumption, area utilization, and timing performance for resource-constrained environments. Existing solutions often struggle to achieve a balance between flexibility and efficiency, highlighting the need for custom designs tailored to specific application requirements. This work builds on the progress made in previous studies by introducing a custom IP that integrates multiple image processing operations into a unified system, optimized for real-time performance on FPGAs.

III. PROPOSED CUSTOM IP

The proposed custom IP is designed specifically for real-time Scharr edge detection, while the Verilog implementation includes additional edge detection methods such as Prewitt, Sobel, Laplacian, and blurring operations. The system processes 512x512 grayscale BMP images directly, eliminating the need for intermediate HEX file conversions and simplifying the workflow. A 3x3 convolution kernel is employed for neighborhood-based processing, with a line buffer efficiently handling pixel data, particularly in blurring and edge detection tasks. The hardware architecture integrates the custom Scharr edge detection IP with a Zynq Processing System (PS) using AXI interconnects for high-bandwidth communication. A DMA controller enhances efficiency by enabling direct memory access, reducing CPU involvement, and ensuring continuous data processing in real time. The workflow includes implementing all core image processing operations in Verilog, packaging only the Scharr edge detection module as a custom IP in Xilinx Vivado, and integrating it with the Zynq PS in a block design. A flowchart, shown in Fig. 1, outlines the workflow, beginning with image input preparation in BMP format and concluding with real-time display or storage of results. This structured approach ensures a seamless hardware-software co-design, optimizing performance and adaptability for real-time image processing.

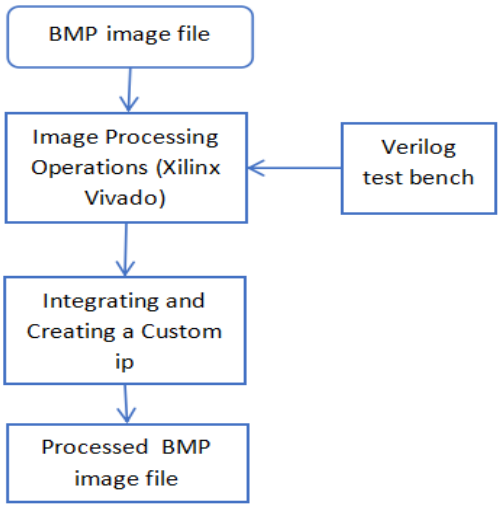

Fig.1 Flow Chart of Proposed Custom IP Design

IV IMPLEMENTATION

The implementation of the proposed system involves designing Verilog modules for image processing operations, packaging them into a custom IP, and integrating the IP into the hardware-software co-design. The system processes grayscale images of 512x512 resolution directly in BMP format. The original input image, as shown in Fig.2, is a 512x512 grayscale BMP image that serves as the baseline for all image processing operations. This image was selected for its standard resolution, which is well-suited for testing the system's capabilities while maintaining compatibility with the custom IP's architecture. The image contains varying intensity levels, making it ideal for demonstrating the effects of blurring, and edge detection.

Fig.2 Original Input Image

A. Blurring

Blurring reduces noise and smoothens the image by averaging pixel values within a 3x3 neighborhood. This operation prepares the image for subsequent edge detection by eliminating high-frequency variations. The simulation in Fig.3 waveform has an input pixel with a hexadecimal value of 6B (decimal 107) produces an output pixel of 92 (decimal 146) after the averaging operation. This change illustrates how box blur reduces sharp contrasts by adjusting pixel intensities towards a local average. The signals in the waveform indicate that the data is processed correctly in a synchronous manner. The output image in Fig.4 shows a softened version of the input image with reduced noise.

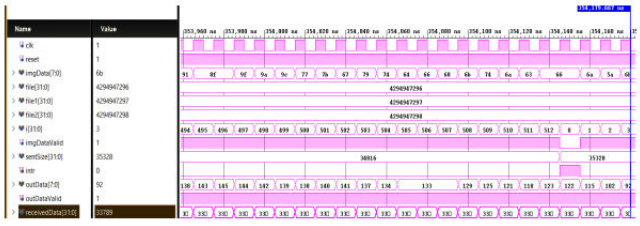

Fig.3 Simulation Result of Blur Operation

Fig.4 Blur Operation Output Image

B. Sobel Edge Detection

Sobel edge detection identifies edges in an image by highlighting areas of rapid intensity change. Fig.5 illustrates the simulation waveform for the Sobel edge detection operation, demonstrating the step-by-step processing of pixel values. Three input files are used for image processing, while two output files are generated one for the processed image file and another for the processed hex file. The imgData signal corresponds to the grayscale pixel values of the original image before edge detection is applied. This simulation verifies the correct implementation of the Sobel filter by effectively detecting edges in the input image. The outData signal outputs values of 0 and 255, representing black and white pixel intensities, respectively, which form the final edge-detected image. Output image in Fig.6 displays clear vertical and horizontal edges. Sobel is efficient for general edge detection but may lack precision for diagonal edges.

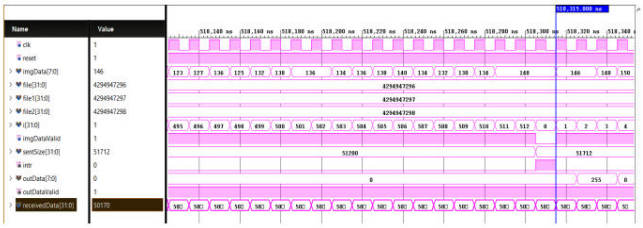

Fig.5 Simulation Result of Sobel Operation

Fig.6 Sobel Operation Output Image

C. Prewitt Edge Detection

Prewitt edge detection works similarly to Sobel but employs a simpler gradient calculation. It is less sensitive to noise but provides comparable edge detection results. Fig.7 presents the simulation waveform for the Prewitt edge detection operation, showcasing the sequential processing of pixel values. The imgData signal represents the grayscale intensity values of the original image before edge detection is applied. The correctness of the Prewitt filter implementation is validated through this simulation. The outData signal produces values of 0 and 255, corresponding to black and white pixel intensities, thereby forming the final edge-detected image. while the output image in Fig.8 demonstrates the edges detected with slightly reduced sensitivity to noise.

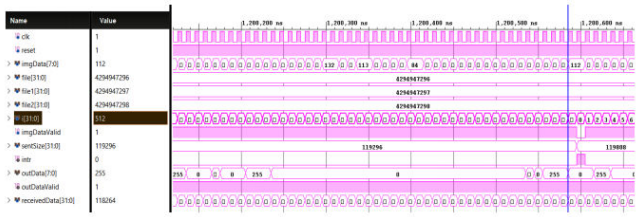

Fig.7 Simulation Result of Prewitt Operation

Fig.8 Prewitt Operation Output Image

D. Scharr Edge Detection

Scharr edge detection offers superior rotational symmetry and is effective in detecting edges in complex images. Fig.9 presents the simulation waveform for the Scharr edge detection operation, showcasing the sequential processing of pixel values. The imgData signal represents the grayscale intensity values of the original image before edge detection is applied. The correctness of the Scharr filter implementation is validated through this simulation. The outData signal produces values of 0 and 255, corresponding to black and white pixel intensities, thereby forming the final edge-detected image. Compared to other operators, the Scharr filter applies an optimized kernel that enhances gradient calculations, resulting in sharper and more distinct edge detection. Output image in Fig.10 shows well-defined edges with high clarity, making it particularly suitable for high-precision applications.

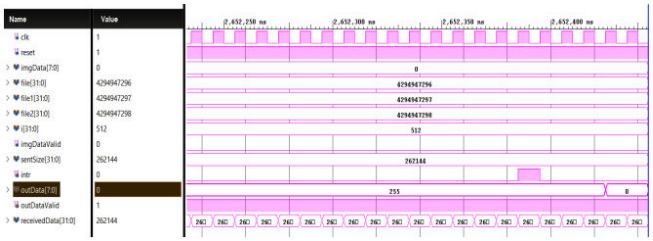

Fig.9 Simulation Result of Scharr Operation

Fig.10 Scharr Operation Output Image

E. Laplacian Edge Detection

Laplacian edge detection uses second-order derivatives to highlight regions of rapid intensity change. This method is more sensitive to fine details but can amplify noise. Fig.11 presents the simulation waveform for the Laplacian edge detection operation, illustrating the pixel-wise processing of the input image. The imgData signal represents the grayscale intensity values of the original image before edge detection is applied. The correctness of the Laplacian filter implementation is validated through this simulation, which calculates the second derivative of pixel intensities to enhance edge detection. The outData signal produces values of 0 and 255, corresponding to black and white pixel intensities, thereby forming the final edge-detected image. Unlike other gradient-based methods, the The output image in Fig.12 reveals intricate edge details, emphasizing the finer features of the input image.

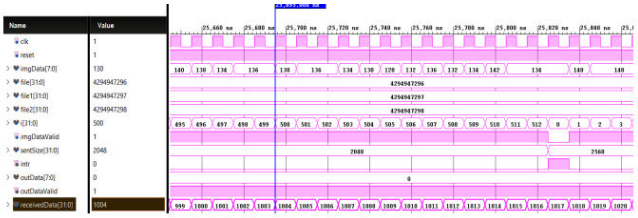

Fig.11 Simulation Result of Laplacian Operation

Fig.12 Laplacian Operation Output Image

IV. PERFORMANCE METRICS

In this work, edge density is calculated by binarizing the edge-detected images and summing the binary values to determine the number of detected edges. The formula for edge density can be expressed as:

Edge Density =

(Number of Edge Pixels / Total Number of Pixels) × 100

Where,

Number of Edge Pixels is the count of pixels detected as edges (typically those with high gradient values after applying an edge detection filter).

Total Number of Pixels is the total number of pixels in the image (Width × Height).

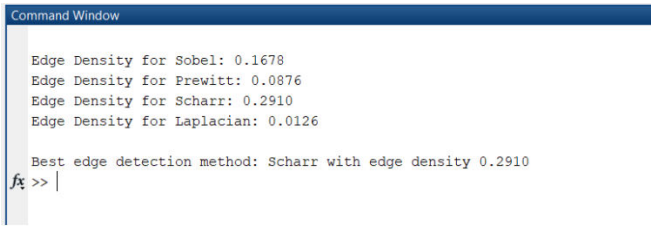

Fig.13 Result for Edge Density

Fig.13 presents the MATLAB results for edge density across the different edge detection techniques. The results clearly indicate that the Scharr edge detection method consistently yields the highest edge density value, followed by the Sobel, Prewitt, and Laplacian methods. Based on this analysis, a custom IP for Scharr edge detection is created to leverage its superior performance in detecting edges with higher accuracy. By implementing the Scharr operator as a dedicated hardware module, the system achieves optimized processing speed and efficiency, making it well-suited for real-time image processing applications.

V HARDWARE SYSTEM DESIGN FOR CUSTOM IP

The Custom IP (imageprocess_0) is illustrated in Fig.14. This design connects the Custom IP to the Zynq Processing System (PS) using AXI interconnects IP and a DMA controller IP, ensuring efficient data transfer and enabling real-time image processing. The Zynq PS, powered by a dual-core ARM processor, serves as the central controller, managing tasks such as configuring the custom IP, handling data flow between external storage and the Programmable Logic (PL), and executing application-level software. The AXI interconnects IP core provide a high-speed communication channel between the PS and the PL, facilitating low-latency transfer of input data and processed results. At the core of the system, the custom IP performs operations like brightness adjustment, inversion, thresholding, blurring, and edge detection in the PL. To further enhance efficiency, a DMA controller IP core is employed to handle direct memory transfers between the system memory and the custom IP. This minimizes CPU involvement in data handling, allowing the processor to focus on higher-level control tasks. The DMA ensures continuous and high-speed streaming of image data, critical for achieving real-time performance.

The block diagram in Fig.14 showcases how the PS, custom IP, DMA controller, and memory are interconnected. Here input gray scale image(barbara.bmp) is transferred by the custom IP via the DMA. The processed image is written back to memory, where they can be accessed for display or storage. This hardware-software co-design leverages the processing capabilities of the FPGA for acceleration while maintaining the flexibility of the ARM processor for system management. The architecture effectively combines parallelism, low-latency data handling, and resource efficiency, making it ideal for high-performance real-time image processing applications.

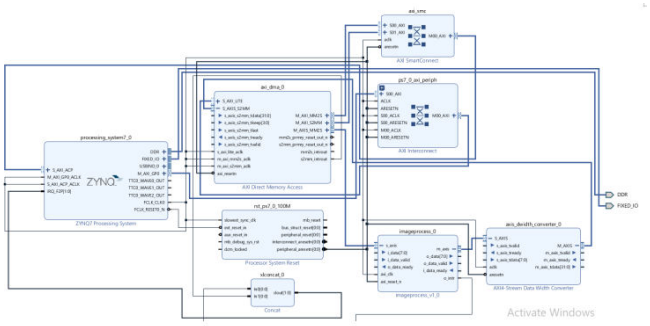

Fig.14 Hardware System Design of Proposed Custom IP System

VI RESULT AND DISCUSSION

Performance metrics underscore the efficiency of the custom IP, with detailed analysis highlighting improvements in power consumption, area utilization, and timing performance. The power analysis, as shown in Fig.15, indicates a reduction in static power consumption.

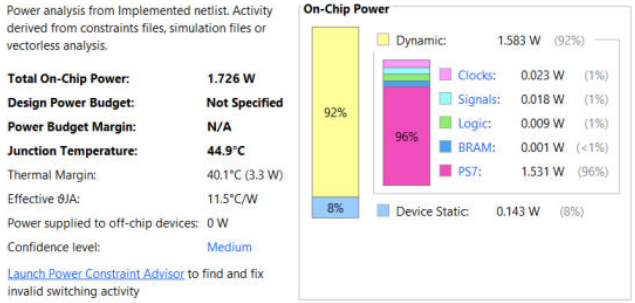

Fig.15 Power Report

The area utilization results, illustrated in Fig.16, demonstrates the optimal use of FPGA resources, ensuring scalability. The compact design of the custom IP balances functionality and resource usage, allowing for effective integration into the Zynq-based architecture.

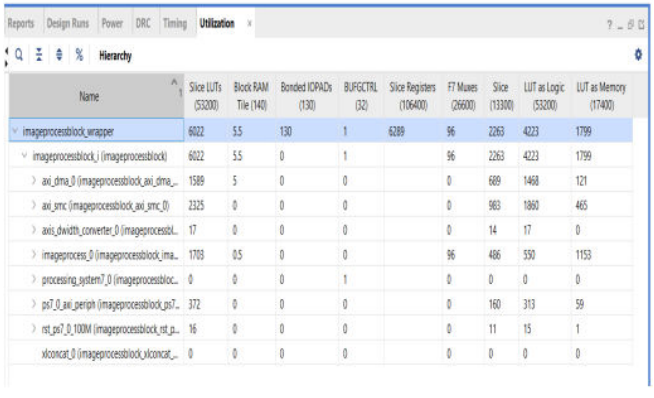

Fig.16 Area Report

The timing analysis, as depicted in Fig.17, confirms that the design meets real-time processing requirements, with a slack time of zero, ensuring optimal performance.

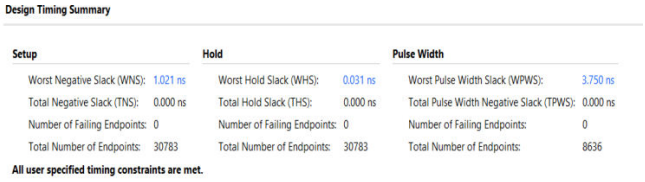

Fig.17 Timing Report

Among edge detection techniques, Scharr achieves the highest edge density, as reflected in the analysis, providing superior sensitivity and accuracy in identifying edges. Visual results for operations like blurring, and edge detection, further validate the system's functionality. The clear and noise-free outputs are achieved through precise kernel operations and efficient data handling. The ability to process BMP images directly simplifies the workflow and enhances system usability, making the design practical for real-world applications. Each metric power, area, and timing demonstrate the system's capability to balance efficiency and performance effectively.

VI CONCLUSION AND FUTURE WORK

This work presents the design and implementation of a Custom IP for Scharr edge detection on Xilinx SoC FPGA, overcoming the limitations of standard IP cores. By developing a dedicated IP for Scharr edge detection, the system achieves enhanced processing efficiency and accuracy, making it well-suited for real-time image processing applications. Performance evaluation in terms of power, area, and timing metrics demonstrates the effectiveness of the proposed approach, highlighting its suitability for high-speed edge detection tasks. The results confirm that FPGA-based solutions provide an optimized and efficient framework for real-time image processing.

Future work will focus on extending the IP to support color images and implementing adaptive processing techniques using machine learning. Additional image processing operations, such as sharpening can also be integrated. By building on the proposed system, this work aims to further advance the field of FPGA-based image processing, catering to emerging demands in automation, healthcare, and security.

REFERENCES

[1] V. S. Surwase and S. N. Pawar, "VLSI implementation of image processing algorithms on FPGA," *International Journal of Electronic and Electrical Engineering*, vol. 3, pp. 139–145, 2010.

[2] Kim, J.H. and Cho, J.D., "A Real-Time 3D Image Refinement Using Two-Line Buffers," in Proceedings of the 13th International Conference on Advanced Communication Technology (ICACT2011), IEEE, pp. 778-781, February 2011.

[3] S. Samanta, S. Paik, S. Gangopadhyay, and A. Chakrabarti, "Processing of image data using FPGA-based microblaze core," *Springer*, *ResearchGate*, pp. 241–246, 2011.

[4] J. M. Maatta, J. Vanne, T. D. Hamalainen, and J. Nikkanen, "Generic software framework for a line-buffer-based image processing pipeline," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1442–1449, 2011.

[5] N. H. J. A. Ahmad and A. Amira, "FPGA-based implementation of 3-D Daubechies for medical image compression," 2019 Third International Conference on Inventive Systems and Control (ICISC), pp. 683–688, 2012.

[6] I. Chiuchisan, "A New FPGA-Based Real-Time Configurable System for Medical Image Processing," in Proceedings of the 4th IEEE International Conference on E-Health and Bioengineering, pp. 1–5, 2013.

[7] N. P. Raut and A. V. Gokhale, "FPGA Implementation for Image Processing Algorithms Using Xilinx System Generator," *Journal of VLSI and Signal Processing*, vol. 2, no. 4, pp. 1–6, Jun. 2013.

[8] C. Wang and S. Zhu, "A design of FPGA-based system for image processing," *Review of Computer Engineering Studies*, vol. 2, pp. 25–30, 2015.

[9] H. H. Al-Ithawi and A. M. M. Abderazek, "Real-Time Image Processing Using Xilinx FPGA," *Journal of Applied Science and Engineering Research*, vol. 4, no. 3, pp. 109-114, 2015.

[10] M. A. Altuncu, T. Guven, Y. Becerikli, and S. Sahin, "Real-Time System Implementation for Image Processing with Hardware/Software Co-Design on the Xilinx Zynq Platform," *International Journal of Information and Electronics Engineering*, vol. 5, no. 6, pp. 466–470, Nov. 2015.

[11] C. S. P. Reddy and P. K. Sahu, "FPGA-Based Implementation of Image Processing Algorithms Using Verilog," *International Journal of Engineering Research & Technology*, vol. 5, no. 5, pp. 712-718, 2016.

[12] A. R. M. Hamzah, F. Mohd-Yusof, and H. A. F. Awang, "High-Performance Image Processing Using Custom IPs on FPGA," *Journal of Real-Time Image Processing*, vol. 14, pp. 1005-1018, 2017.

[13] C. O. Ancuti, C. Ancuti, C. De Vleeschouwer, and P. Bekaert, "Color Balance and Fusion for Underwater Image Enhancement," *IEEE Transactions on Image Processing*, vol. 27, no. 1, pp. 379–393, Jan. 2018.

[14] A. A. Azam, S. F. Khan, and A. Z. Qamar, "Design and Implementation of Image Processing Algorithms Using Zedboard," *International Journal of Computer Applications*, vol. 179, no. 40, pp. 1-6, 2018.

[15] A. HajiRassouliha, A. J. Taberner, M. P. Nash, and P. M. F. Nielsen, "Hardware implementation of image processing algorithms on FPGA," *Elsevier*, *Signal Processing Image Communication*, vol. 68, pp. 35-42, 2018.

[16] R. K. Ranjan, M. Bharti, and R. Kumar, "Implementation of Image Processing Applications on Zynq-7000 SoC," *International Journal of Engineering Research and Applications*, vol. 9, no. 5, pp. 51-58, 2019.

[17] K. P. Anjaneyulu and D. R. Patil, "A Novel Image Processing Framework for FPGA-Based Implementation," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 6, pp. 529-532, 2019.

[18] D. A. Devi, T. S. Savithri, and S. Sugun.L, "Design and implementation of real time data acquisition system using reconfigurable soc," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 9, pp. 325–331, 2020.

[19] D. Lee and Y. Kim, "FPGA-Based Implementation of Image Filtering using 3x3 Kernel with AXI Interface in Vivado," *Journal of Imaging Systems and Technology*, vol. 56, no. 3, pp. 125–134, 2020.

[20] G. Alaverdian, A. Mkrtchyan, M. Minasyan, and D. Akopyan, "Image Processing on FPGAs: Design and Implementation with Vivado HLS," *International Journal of Electronics and Communications*, vol. 118, pp. 35–42, Jan. 2021.

[21] D. A. Devi, N. R. Kathula, G. Kalluri, and L. S. Bondalapati, "Design and implementation of image processing application with zynq SoC," *International Journal of Computing and Digital Systems*, vol. 12, pp. 1–10, 2023.

[22] https://www.hlevkin.com/hlevkin/06testimages.htm