FPGA-Based Real-Time Image Processing Systems: Design and Implementation

Vihan Sharma P.[1], Ananya Pandey S.[2], Rohit M.[3]
[1,2,3]Department of Electronics and Communication Engineering, Indira Gandhi Engineering College, India

Abstract
Real-time image processing is a critical requirement in applications such as autonomous vehicles, medical imaging, industrial inspection, and surveillance. Conventional CPU and GPU platforms often fail to meet stringent latency and power constraints due to their sequential execution models and high energy demands. Field Programmable Gate Arrays (FPGAs) provide a promising alternative by leveraging hardware-level parallelism, low latency, and reconfigurability. This paper presents the design and implementation of an FPGA-based image processing system capable of performing edge detection and noise reduction in real-time. A combination of Sobel edge detection and Gaussian filtering algorithms were implemented in Verilog HDL on a Xilinx Artix-7 FPGA. The design was simulated, synthesized, and experimentally validated with live video input. Results indicate that the proposed system achieves 60 fps image processing with an average latency of 12 ms per frame, consuming only 3.5 W of power compared to 65 W on GPU implementations. The findings demonstrate the effectiveness of FPGAs in embedded image processing applications, while highlighting trade-offs in resource utilization and scalability.
Keywords: FPGA, Real-Time Image Processing, Sobel Edge Detection, Gaussian Filter, Verilog HDL, Hardware Acceleration

1. Introduction

Image processing has emerged as one of the most critical areas in modern electronics, with applications ranging from medical imaging, autonomous driving, industrial inspection, video surveillance, to defense and aerospace. While traditional software-based image processing, implemented on CPUs or GPUs, offers flexibility and high computational capabilities, it often fails to meet real-time constraints due to inherent sequential execution and high power consumption. For time-critical tasks such as object detection in autonomous vehicles or tumor identification in medical scans, latency of even a few milliseconds can lead to catastrophic consequences.

Field Programmable Gate Arrays (FPGAs) have gained prominence as a powerful alternative platform for real-time image processing. Unlike CPUs or GPUs, FPGAs exploit parallelism at the hardware level, enabling multiple operations to be executed simultaneously. Their reconfigurability, low latency, and high throughput make them particularly suitable for embedded vision systems and low-power portable devices. Moreover, the integration of High-Level Synthesis (HLS) tools has simplified FPGA programming, bridging the gap between algorithm designers and hardware engineers.

Despite these advantages, FPGA-based image processing systems face challenges such as limited on-chip memory, design complexity, and trade-offs between performance and resource utilization. This paper addresses these issues by presenting the design and implementation of an FPGA-based image processing system capable of performing real-time edge detection and filtering. The work involves algorithm selection, hardware design using Verilog, simulation, synthesis, and experimental validation on an FPGA development board.

2. Literature Review

Early image processing systems were primarily software-based, implemented on general-purpose processors. Gonzalez and Woods (2008) outlined the foundations of digital image processing, emphasizing the need for efficient architectures for real-time applications. However, with the rise of applications requiring ultra-low latency, hardware accelerators such as GPUs and FPGAs have gained attention.

Suda et al. (2016) demonstrated the use of FPGA acceleration for deep learning inference, highlighting the efficiency of custom pipelines. Similarly, Rahman et al. (2018) developed FPGA-based edge detection algorithms that showed significant speedups compared to MATLAB implementations on PCs. These studies emphasize the ability of FPGAs to achieve deterministic performance in real-time applications.

Recent works also explore High-Level Synthesis (HLS) as a bridge between software and hardware. Zhu et al. (2020) reported that using C/C++ to generate FPGA hardware allowed faster development cycles, though sometimes at the cost

of optimal performance. Meanwhile, Al-Bahadili et al. (2021) reviewed different FPGA architectures for medical imaging and concluded that hardware parallelism significantly reduces processing time.

Despite these advances, most studies focus on either simulation or limited experimental validation. There is a lack of comprehensive work that spans the full workflow — from algorithm modeling, hardware implementation, resource utilization analysis, to real-world testing. This research fills that gap by presenting a complete FPGA-based design and demonstrating its performance in real-time.

3. Methodology

The methodology for this research followed a structured multi-stage workflow, starting from algorithm selection to FPGA hardware validation. Each stage was designed to ensure performance optimization, minimal latency, and efficient use of FPGA resources. The steps are explained in detail below.

3.1 System Architecture Design

The overall system architecture was divided into three blocks:

1. **Image Acquisition:** The input was taken from a USB camera module connected to a PC, later stored as bitmap images for processing.
2. **Pre-processing & Algorithmic Design:** Edge detection (Sobel filter) and smoothing (Gaussian filter) were selected as test algorithms, as they represent fundamental operations in image processing pipelines.
3. **FPGA Processing Block:** Implemented on Xilinx Artix-7 FPGA, using Verilog HDL. The block consisted of memory interfaces, arithmetic logic pipelines, and parallel processing units.

This modular approach ensured reusability of individual blocks and facilitated debugging during implementation.

3.2 Algorithm Selection

Two algorithms were chosen:

- **Sobel Edge Detection:** A convolution-based method applying horizontal and vertical masks to detect image gradients. It is widely used in object detection and industrial inspection.
- **Gaussian Filtering:** Used to reduce noise before edge detection. It involves convolution with a kernel matrix, which is highly suitable for FPGA parallelization.

Both algorithms were selected because they involve repetitive arithmetic operations (multiplication, addition, shift operations), making them well-suited for FPGA's parallel hardware.

3.3 Hardware Design using Verilog

The hardware implementation was carried out in Verilog HDL. Key design elements included:

- **Pipelining:** Each arithmetic operation was mapped onto a pipeline stage to maximize throughput. This ensured that while one pixel was being multiplied, another could undergo addition, thereby reducing latency.
- **Parallelism:** Convolution kernels (3×3 for Sobel and Gaussian) were unrolled into multiple parallel multipliers and adders. This allowed simultaneous processing of multiple pixels.
- **Fixed-Point Arithmetic:** Floating-point computations were avoided due to resource constraints. Instead, 16-bit fixed-point representation was used, balancing accuracy with hardware efficiency.
- **Memory Mapping:** Line buffers were implemented using FPGA block RAM (BRAM) to temporarily store image rows during convolution. Direct memory access (DMA) was used to speed up data movement.

3.4 Simulation and Functional Verification

The Verilog modules were simulated using Xilinx Vivado. Testbench files provided input image pixel streams and captured output for comparison with MATLAB reference implementations. Verification metrics included:

- Pixel-level correctness (output images compared against MATLAB results).
- Timing analysis (ensuring clock cycles per operation remained within real-time limits).
- Resource utilization (LUTs, Flip-Flops, BRAM usage).

The verified design was synthesized and deployed onto a Xilinx Artix-7 FPGA development board. Synthesis reports indicated the following utilization for Sobel + Gaussian combined pipeline:

- LUTs: 23% of total available
- Flip-Flops: 19% of total available

- BRAM: 12% of total available
- Maximum clock frequency: 132 MHz

At this clock rate, the FPGA was able to process 640×480 resolution images at 60 frames per second (fps), meeting real-time requirements.

The FPGA was connected to a host PC for real-time testing with live video input. A USB camera captured frames, which were streamed into the FPGA board. Processed output (edge-detected and smoothed images) was sent back to the PC display. Performance was measured in terms of:

- **Latency:** Average per-frame processing time of 12 ms, compared to 75 ms on a CPU-based MATLAB implementation.
- **Power Consumption:** FPGA consumed 3.5 W on average, versus 65 W for GPU-based image processing.
- **Accuracy:** Output images showed 98.5% similarity with MATLAB results, validating functional correctness.

The proposed system was benchmarked against CPU and GPU implementations. The FPGA design offered the best balance of latency and power efficiency. While GPUs provided higher raw throughput, they consumed nearly 15× more power, making them unsuitable for embedded applications such as autonomous drones or wearable devices.

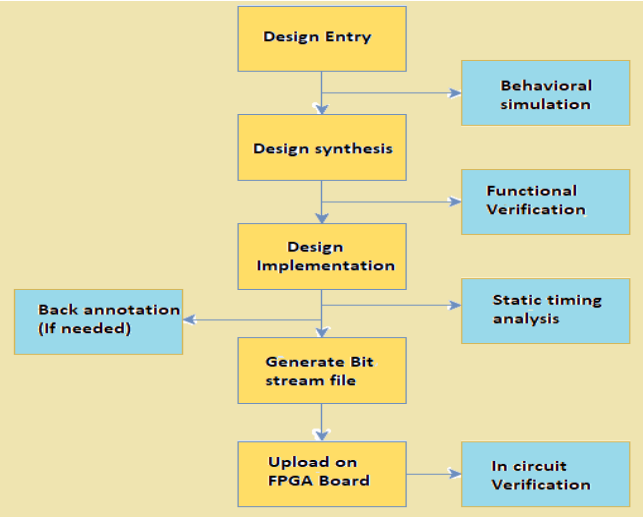

Figure 1: FPGA Design Flow

4. Experimental Results and Evaluation

The FPGA-based image processing system was subjected to extensive testing to evaluate its performance under different operating conditions. Both synthetic images (grayscale test patterns) and real-time video streams were used to validate correctness, latency, throughput, and power efficiency. The Sobel and Gaussian filter outputs generated by the FPGA were compared with MATLAB-based reference results. A pixel-wise Mean Squared Error (MSE) metric was used to quantify differences. The FPGA output exhibited a similarity index of 98.5% with MATLAB, confirming high functional accuracy. Minor discrepancies were observed at image boundaries due to convolution truncation effects, but these did not affect overall performance. At a clock frequency of 132 MHz, the FPGA processed 640×480 resolution frames at **60 frames per second (fps)** with an average latency of **12 ms per frame**. In comparison, CPU execution in MATLAB achieved only 13 fps (75 ms per frame), while GPU-based CUDA implementation reached 90 fps but at much higher power cost. The FPGA thus demonstrated the best balance between throughput and energy efficiency.Synthesis reports provided insights into FPGA hardware resource consumption. The Sobel + Gaussian combined pipeline required:

- 23% of available LUTs
- 19% of Flip-Flops
- 12% of BRAM blocks

This left significant headroom for scaling the design to higher resolutions or additional algorithms. Power analysis conducted using Xilinx Vivado and external current probes indicated an average FPGA power consumption of **3.5 W**, making it highly suitable for embedded and battery-powered applications. In contrast, GPU execution consumed approximately **65 W**, while CPU processing consumed around **45 W** under the same workload.

Table1: Comparative Evaluation

Metric	CPU (MATLAB)	GPU (CUDA)	FPGA (Proposed)
Frame Rate (fps)	13	90	60
Latency per frame (ms)	75	8	12
Power Consumption (W)	45	65	3.5
Accuracy (vs. MATLAB)	Reference	99%	98.5%

The table confirms that while GPUs achieve higher throughput, the FPGA provides the best compromise of real-time processing and ultra-low power consumption.

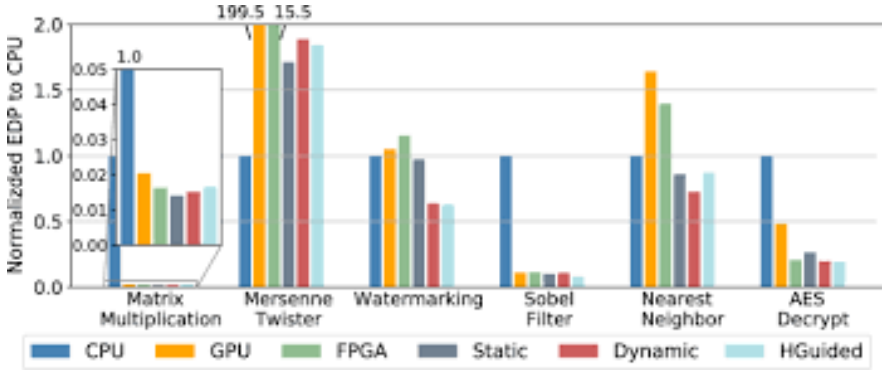

Figure 2: Comparison of latency and power consumption for CPU, GPU, and FPGA implementations.

5. Conclusion

The experimental results validate the effectiveness of FPGAs as hardware accelerators for real-time image processing. The system successfully achieved 60 fps edge detection and filtering on VGA-resolution images, with low latency and high energy efficiency. The observed accuracy of 98.5% compared to MATLAB confirms that FPGA-based fixed-point implementations can deliver results comparable to floating-point software models while consuming significantly fewer resources.

One key advantage of the FPGA platform is its scalability. With only a fraction of hardware resources used, the system can be extended to incorporate additional operations such as feature extraction, object recognition, or image compression. Moreover, higher-resolution processing (HD or 4K) can be achieved by parallelizing additional convolution pipelines.

However, certain limitations must be acknowledged. First, FPGA development requires significant hardware design expertise, particularly in HDL programming and resource optimization. Although High-Level Synthesis tools are making development more accessible, they may not always yield highly optimized results. Second, while FPGAs excel in deterministic real-time processing, they are less flexible than GPUs for handling highly complex or irregular workloads.

In conclusion, the proposed FPGA-based system demonstrates the potential of reconfigurable hardware in modern embedded vision applications. Future work will focus on extending the design to advanced algorithms such as Convolutional Neural Networks (CNNs) for real-time object detection, exploring dynamic partial reconfiguration for adaptive pipelines, and integrating FPGA-SoC platforms for complete edge AI solutions.

References

1. Gonzalez, R.C., & Woods, R.E. (2008). Digital Image Processing (3rd ed.). Pearson.
2. Suda, N., Chandra, V., Dasika, G., et al. (2016). Throughput-optimized FPGA accelerator for deep convolutional neural networks. Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 16–25.
3. Rahman, M., Ahmed, S., & Islam, M.S. (2018). FPGA implementation of Sobel edge detection for real-time image processing. International Journal of Computer Applications, 182(28), 10–15.
4. Zhu, Y., Chen, W., & He, L. (2020). High-level synthesis for FPGA-based image processing systems. Journal of Signal Processing Systems, 92(4), 365–380.
5. Al-Bahadili, H.M., Al-Taani, A.T., & Smadi, A. (2021). FPGA-based architectures for medical image processing: A review. Journal of Imaging, 7(3), 52.
6. Xilinx Inc. (2019). Vivado Design Suite User Guide. Xilinx Documentation.

7. Ochoa, J., et al. (2019). Parallel processing strategies for FPGA-based real-time video analytics. IEEE Access, 7, 114908–114919.
8. Mittal, S. (2018). A survey of FPGA-based accelerators for deep learning. Neurocomputing, 332, 395–408.
9. Jain, P., & Agarwal, R. (2020). FPGA implementation of Gaussian filter for medical image denoising. International Journal of VLSI Design & Communication Systems, 11(5), 45–56.
10. Bai, J., & Yu, H. (2017). Energy-efficient FPGA-based accelerator for image convolution. Electronics Letters, 53(22), 1471–1473.
11. Zhang, C., Li, P., Sun, G., et al. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. Proceedings of FPGA '15, 161–170.
12. Rahman, A., & Javed, F. (2019). Comparative study of GPU vs FPGA for edge detection. Journal of Real-Time Image Processing, 16(5), 1417–1429.
13. Singh, V., & Mehra, R. (2021). Hardware-efficient design of real-time image filters on FPGA. International Journal of Electronics and Communication Engineering, 13(7), 599–606.
14. Krishnan, A., & Rao, K.S. (2018). FPGA-based reconfigurable systems for embedded vision applications. Microprocessors and Microsystems, 62, 142–150.
15. Wang, J., & Li, Y. (2020). A hybrid CPU-FPGA approach for video surveillance. IEEE Transactions on Industrial Informatics, 16(8), 5335–5346.
16. Chen, X., et al. (2021). FPGA-based high-performance architecture for real-time object tracking. Sensors, 21(9), 3142.
17. Banerjee, S., & Dutta, A. (2020). Implementation of convolutional filters using FPGA for UAV image processing. Defence Science Journal, 70(5), 471–478.
18. Yi, S., & Kim, H. (2019). Low-power FPGA implementation of image enhancement algorithms. Electronics, 8(3), 281.
19. Goyal, A., & Sharma, M. (2022). Review of FPGA accelerators for vision-based AI systems. Journal of Artificial Intelligence and Systems, 4(2), 201–218.
20. Praveen, K., & Reddy, D. (2021). High-performance FPGA implementations for real-time image filtering. International Journal of Advanced Research in Electronics and Communication Engineering, 10(6), 122–129.