computers

MDPI

Article

FPGA Modular Scalability Framework for Real-Time Noise Reduction in Images

Ng Boon Khai, Norfadila Mahrom, Rafikha Aliana A. Raof, Teo Sje Yin and Phaklen Ehkan *

Faculty of Intelligent Computing, University Malaysia Perlis, Arau 02600, Perlis, Malaysia; nbkhai@studentmail.unimap.edu.my (N.B.K.); norfadila@unimap.edu.my (N.M.); rafikha@unimap.edu.my (R.A.A.R.); sjeyin@studentmail.unimap.edu.my (T.S.Y.)
* Correspondence: phaklen@unimap.edu.my

Abstract

Image noise degrades image quality in applications such as medical imaging, surveillance, and remote sensing, where real-time processing and high accuracy are critical. Software-based denoising can be flexible, but often suffers from latency and low throughput when deployed on embedded or edge systems. A Field Programmable Gate Array (FPGA)-based system offers parallelism and lower latency, but the existing work typically focusses on fixed architectures rather than scalable framework supporting multiple filter models. This paper presents a high-performance, resource-efficient FPGA-based framework for real-time noise reduction. The modular, pipelined architecture integrates median and adaptive filters, managed by a state machine-based control unit to enhance processing efficiency. Implemented on a Cyclone V FPGA using Quartus Prime 22.1std, the system provides scalability through adjustable Random Access Memory (RAM) and supports multiple denoising algorithms. Tested on Leena images with salt-and-pepper noise, it processes 10% noise in 1.724 ms in a simulated environment running at 800 MHz; it was compared with Python version 3.11.2 with the OpenCV-library version 4.8.076 on a general-purpose Central Processing Unit (CPU) (0.0201 ms). The proposed solution demonstrates low latency and high throughput, making it well-suited for embedded and edge computing applications.

Keywords: FPGA; modular FPGA architecture; scalable pipeline framework; hardware-based filtering framework; noise filtering; salt-and-pepper noise reduction

1. Introduction

In digital image processing, noise reduction is a fundamental task for enhancing image clarity and quality, particularly in applications where precision is critical, such as medical imaging, satellite imaging, and security systems. During image acquisition and transmission, various types of noise may be introduced [1], compromising the fidelity of the captured images. Among these, salt-and-pepper noise represents a fixed-valued random noise type that significantly disrupts pixel intensities [2,3]. In fixed-valued noise, pixels are corrupted with extreme values, while in random-valued noise, pixel intensities are scattered across a range. Both noise types can severely degrade important image details [4].

Effective noise reduction is essential to ensure that images retain informative content, accurate analysis, and interpretation without removing critical features. Conventional methods such as Gaussian and median filtering have been widely applied to suppress image noise [5]. Gaussian filters are particularly effective for smoothing and attenuating

Check for updates

Academic Editor: Paolo Bellavista

Received: 25 October 2025
Revised: 25 November 2025
Accepted: 9 December 2025
Published: 1 January 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Gaussian noise, while median filters excel in mitigating impulse noise by replacing corrupted pixels with the median values of their neighbours [6,7]. However, these approaches tend to blur images and diminish important features when noise density is high. Recent developments in adaptive filtering integrate noise detection algorithms that selectively filter only corrupted pixels, unlike traditional full-frame methods. This selective approach improves computational load while preserving the integrity of uncorrupted regions, making it a more efficient strategy for impulse noise suppression [8,9]. Nonetheless, adaptive methods remain computationally demanding in terms of both processing power and memory, limiting their applicability in real-time systems [4].

Field-Programmable Gate Arrays (FPGAs) have emerged as a powerful platform for accelerating complex image processing algorithms due to their inherent parallelism and low-latency characteristics [10]. These advantages make FPGAs highly effective for real-time noise suppression, enabling enhanced performance at reduced hardware costs. However, FPGA-based implementations often rely on custom Verilog designs without standardized frameworks [11]. This practice leads to inconsistencies, inefficiencies, and prolonged development cycles, hindering scalability and portability [12,13].

Despite numerous FPGA-based image denoising implementations, most existing designs are algorithm-specific and lack a standardized, modular structure that supports multiple filtering techniques. Conventional approaches, such as fixed-function median or adaptive filters, often require complete architectural redesigns when integrating new algorithms or modifying image resolutions, thereby limiting reusability, scalability, and development efficiency. Furthermore, few studies have addressed the challenge of creating filter-agnostic and reconfigurable FPGA frameworks capable of adapting to diverse image noise characteristics while maintaining real-time performance and low power consumption.

To bridge this gap, the present work proposes a modular, scalable, and reconfigurable FPGA-based framework that unifies noise reduction architectures under a single hardware platform. Unlike prior FPGA implementations, the proposed design allows plug-and-play integration of different denoising algorithms such as median, adaptive, hybrid, or AI-assisted filters without the need for major architectural modification. This flexibility ensures efficient deployment across varying image sizes, resolutions, and application domains, from embedded imaging to edge computing environments.

This paper introduces a FPGA-based framework designed to overcome the lack of structured approaches for implementing and deploying diverse noise filtering models. The framework emphasizes flexibility and scalability, providing configurable options for impulse noise reduction that can be tailored to application-specific requirements. By establishing a common foundation for FPGA-based image denoising, the framework accelerates development cycles and fosters reusability in both digital and hardware-based real-time image processing. The proposed architecture is modular and filter-agnostic, supporting seamless integration of multiple denoising algorithms including median, adaptive, hybrid, and AI-driven methods without requiring fundamental redesign of the hardware architecture. Scalability is achieved through configurable RAM blocks and modular subcomponents, ensuring adaptability to varying image sizes and resolutions.

Compared to prior FPGA noise-reduction implementations, which are typically algorithm-specific, the proposed framework offers superior consistency and low latency, making it highly suitable for real-time embedded and edge applications. Beyond traditional median filtering, the framework establishes a general-purpose foundation for future integration of advanced denoising techniques, including bilateral filters, wavelet-based methods, and lightweight convolutional neural networks (CNNs).

2. Analysis of Existing Image Filters

2.1. Previous Work

The mentioned implementations form the basis that future work can use to further explore FPGA-based noise reduction systems and offer distinct strategies and improvements to the set methodologies. The distinctive attribute of the system in [14] is its implementation of the decision-based adaptive median (DBAM) filter, which has the remarkable ability to eliminate more than 99% of high-density impulsive noise while maintaining critical image features such as edges and fine details. This design works on an FPGA system at high speed of 254 MHz, making it efficient for real-time application including imaging systems for doctors where precision and speed are critical. Speedy de-noising of images is made possible by the high processing rate which guarantees timely diagnostics and interventions. On the other hand, decision-based filtering relies on some form of information which adds an extra burden on determining the filtering window size and method, which may prolong the processing time, particularly with extreme noise density. This burden requires attention if optimization and efficiency need to be balanced with delicate calibration and fine-tuning [15]. Furthermore, as it was mainly designed with impulse noise in mind, this method's counter action with other noise types such as Gaussian or speckle may not be as effective, emphasizing the need for alternative or combined strategies for broader scope applicability [16]. The narrow focus of the approach demonstrates the necessity of creating multifunctional FPGA-based implementations designed to deal with diverse types of noise without compromising the performance.

Matsubara et al. [15] demonstrate an approach featuring low complexity for noise removal by combining noise detection and filtering using the conditional median technique, focusing on efficient implementation at the FPGA level from a power consumption and area standpoint. This strategy is beneficial for portable and embedded systems due to the limited power and hardware resources available. The compact design enhances usability in resource-limited settings and outperforms standard median filters in image quality preservation, particularly in the presence of impulse noise. However, due to the method's simplicity, unnecessary complexities in detail preservation may arise at higher noise densities where more adaptive techniques could dynamically adjust [17]. Furthermore, because the technique is designed primarily for impulse noise, the approach's inflexibility in dealing with other types of noise, such as Gaussian or speckle noise, calls for less adaptive and more comprehensive FPGA-based noise reduction frameworks.

Jeon et al. [17] focus on the design of an image denoising hardware architecture with real-time processing capabilities aimed at achieving balanced resource utilization and noise reduction efficiency through parallel processing on the FPGA. This system is ideal for rapid image processing tasks in fields like video surveillance or real-time medical diagnosis, where fast and accurate results are essential. Processing speed allows for rapid image analysis to be conducted without delay, which is critical in fast-paced decision-making environments. Yet, as the system works with larger images or noise density increases, the efficiency of processing resources may become strained as higher resource expenditures are needed. Furthermore, the optimized design is effective for denoising, but it is a double-edged sword when broader image processing flexibility is concerned; thus, its range of situational applicability is restricted. These factors illustrate the need for more versatile adaptable designs of architectures built in FPGAs and configured to scale efficiently with processing requirements and more parameters.

The technique proposed by Oliveira et al. [16] tackles noise filtering asynchronously and implements silos of FPGAs that yield high results with low latency and power spending. This technique is highly advantageous in situations where system performance-to-energy usage ratios are critical, such as embedded systems where minimal space is available.

The asynchronous design provides for swifter processing, which increases overall speed and responsiveness. On the contrary, designing and debugging asynchronous circuits can prove to be highly complex, which may add to the development time and optimization effort. The intricate nature of asynchronous circuits requires careful design and testing to ensure reliability and performance. In addition, the focus on asynchronous operation, which is good for some performance metrics, may limit the flexibility and adaptability of the system, particularly when implementing additional processing functions or modifying the design for other uses, indicating the need for more flexible solutions. Performance versus flexibility is one of the key challenges in designing these systems [18].

Thus, such solutions indicate compromises between achieving low complexity, high performance, and flexible application in denoising systems using FPGAs. While some methods are proficient in keeping the quality of the image and running well under specific conditions, others ensure broader application or improve scalability. However, there remains a need for designs that can dynamically adapt to different noise environments and image characteristics, with stable and adaptive operation in a wide range of applications. Future research could study more reconfigurable and adaptive architectures, optimize resource management, and include more sophisticated noise treatment functionalities to address these challenges. This comprehensive approach can help bridge the gap between the cutting-edge and the diversity of actual-world requirements, leading to more universally applicable and more efficient FPGA-based denoising solutions [19]. By focusing on flexibility and scalability, future research can maximize the versatility and effectiveness of FPGA-based systems to enable them to be suitable for an increased number of image processing tasks and environments [20].

2.2. Median Filter

A simple digital processing approach, namely the median filter, is appropriate for, or well-suited to, the removal of impulse noise, also known as "salt-and-pepper noise." Impulse noise is visually apparent in an image as random picture elements with strongly amplified or reduced intensity distributed irregularly around the picture, especially obscuring important details in an imaging process [4]. While linear filters may affect edges in an image, reducing their clarity, the median filter uses a non-linear approach to keep edges intact while efficiently removing noise [21,22]. Apart from traditional approaches, second-order optimization algorithms, as well as deep learning algorithms, have appeared in more sophisticated applications, such as second-order optimization algorithms in denoising remote sensing images contaminated with mixed noise, showing enhanced robustness compared with first-order optimization algorithms [23]. For imaging applications in the medical sector, a method for impulsive noise removal in images, capable of functioning in high impulsive noise density conditions with preserved imaging fidelity, was proposed for real-time processing [24]. Attention-equipped deep networks have demonstrated robust generalization capacities for blind denoising in practical scenarios [25].

The operation of the median filter involves a series of main steps. Initially, a mask is defined, typically a 3×3 matrix, that progresses through the picture pixel by pixel methodically. The mask dictates the pixel neighbourhood to be utilized for filtering. In all masks, the pixel values are sorted in increasing order [26]. For instance, when the mask covers pixels with values 10, 11, 14, 20, 22, 23, 25, 15, and 16 as highlighted with the orange box, they are sorted to be 10, 11, 14, 15, 16, 20, 22, 23, and 25. Sorting the values allows the filter to identify the median value, which is the middle value in the sorted list, as shown in Figure 1.

10	11	14	15	12
20	22	23	21	22
25	15	16	12	14
22	23	21	27	25
24	13	42	12	34

Figure 1. Example pixels of an image.

The median value, in this case 16, is then used to replace the centre pixel of the mask. Substitution is a necessary process as the new pixel value is a good approximation of the intensities of the neighbouring pixels, thereby preventing extreme noise values without averaging them out, which would otherwise blur the edge. By employing the median over the mean, the filter maintains edge and high-frequency detail integrity, yielding improved image quality and clarity [27,28].

Median filtering can be used in software as well as hardware, with each application having some advantages and disadvantages. In software, MATLAB version R2013a is one such platform that provides sample flexibility in testing and prototyping denoising algorithms [29]. This flexibility makes it easier for researchers and developers to test different filter settings and parameters and to re-tune the performance of the filter for specific applications. The robust computational ability of MATLAB allows for the simulation of many scenarios and thus algorithm tuning prior to implementation. In terms of hardware, FPGAs offer a high-performance platform for running median filters, particularly in systems that have real-time needs and where efficiency and speed are critical factors [14,30]. FPGAs have dedicated hardware resources that support high-speed parallel processing and therefore, are very capable of conducting complex filtering operations with minimal latency.

The ability of FPGAs to be programmed to run specific operations with high efficiency allows applications to implement optimized solutions to meet the requirements of real-time image processing. This article talks about a scalable image noise reduction framework in image processing based on an FPGA design with reference to the benefits of FPGA-based implementation to attain real-time performance and improved processing efficiency [31–34].

Thus, the ability of the median filter to preserve image details while effectively eliminating noise makes it an effective instrument in most image processing tasks. Its versatility both in hardware and software realizations further guarantees its utility, providing a robust solution for preserving image quality in various contexts [3,35].

3. Proposed Architecture of FPGA Implementation of Filter

The FPGA-based noise reduction architecture is organized into a set of major functional blocks, each responsible for efficient data handling and processing. Together, these blocks form an integrated system that achieves effective image denoising, as illustrated in Figure 2. The design has been validated using a 512 × 512-pixel image, demonstrating its feasibility for small-scale images. Due to its flexible and reconfigurable structure, the framework can be scaled to handle higher-resolution images, highlighting its adaptability for broader applications.

The Input Image RAM serves as the initial storage for the image data to be processed. Pixel values are preloaded into this RAM, which can be accessed via an address bus (i_ram_addr_ori) and a data output bus (o_ram_data_ori). This enables efficient access to pixel data during noise reduction. Processing speed depends on the FPGA operating

frequency; simulations were carried out at 166 MHz, 500 MHz, and 800 MHz to evaluate performance under different operating conditions.

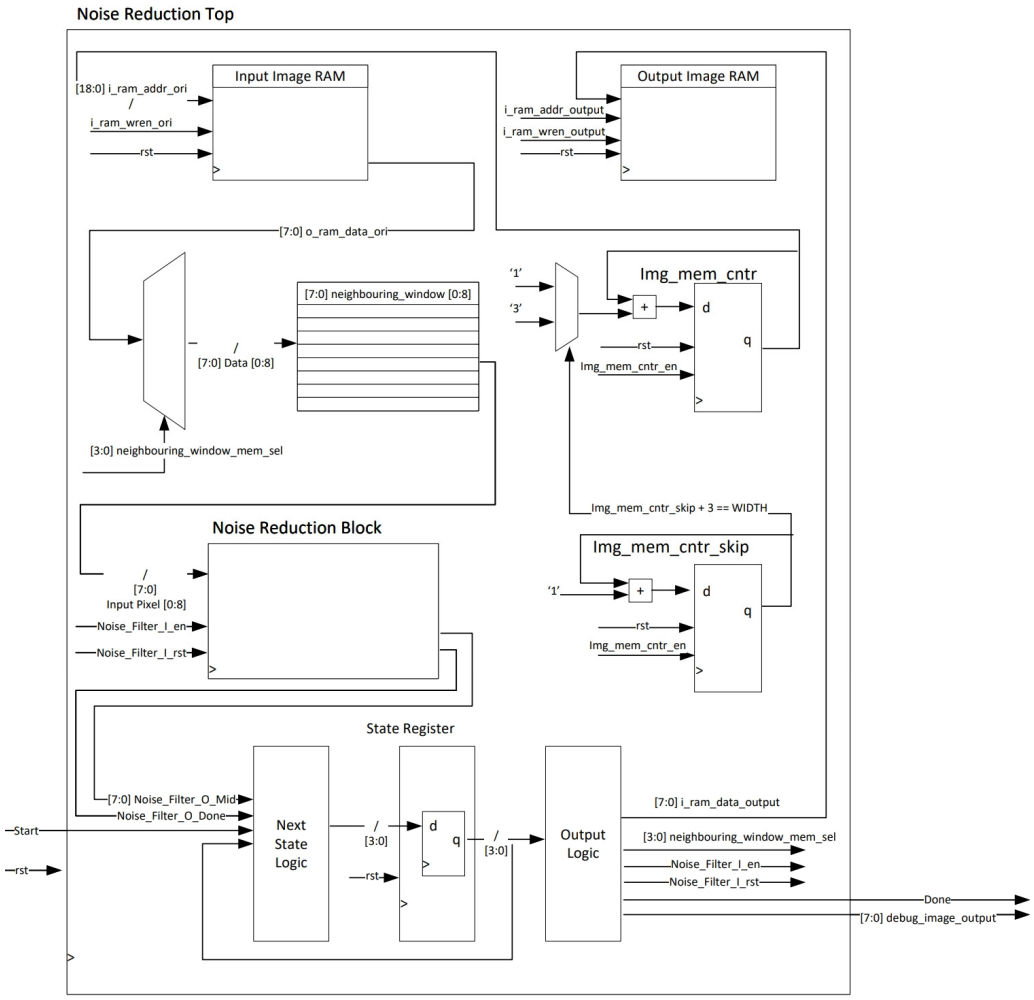


Figure 2. Functional block diagram of the noise filtering design.

At the output stage, the Output Image RAM stores the denoised image data after processing. Similar to the input RAM, it includes an address bus (i_ram_addr_output) and a data input bus (i_ram_data_output). This block ensures smooth transfer of processed pixel values from the noise reduction unit to storage, thereby preserving the integrity of the denoised image.

The Noise Reduction Block is the computational core of the system, where filtering algorithms are executed. In the present implementation, a median filter is employed, using a bubble sort mechanism to rank pixel intensities within a local window. The sorted values enable proper replacement of noisy pixels, improving image quality. Within this block, the State Register tracks the current processing state, coordinating the sequence of operations and ensuring smooth transitions between phases. The Output Logic formats and transfers the final denoised pixel values to the Output Image RAM, maintaining data consistency and validity.

The Neighbouring Window Selector plays a central role in spatial noise reduction by selecting a pixel. Controlled by the signal (neighbouring_window_mem_sel), this module enables context-aware filtering by analysing and modifying pixel values based on their surrounding environment.

To manage pixel sequencing, the Image Memory Counter (img_mem_cntr) controls the addressing and systematic reading of pixels from the Input Image RAM. It uses input

(d) and output (q) signals to synchronize the processing pipeline and ensure sequential handling of all image pixels. Complementing this is the Image Memory Counter Skip (img_mem_cntr_skip), which allows selective skipping of data points or regions. This mechanism can optimize performance by bypassing noise-free regions or handling edge cases, further enhancing processing efficiency.

Inter-component communication is facilitated by registers, while system monitoring and control are handled by the Control and Debug Interface. This interface incorporates control signals, next state logic, and output logic that regulate data flow and sequencing. In addition, debug outputs provide real-time visibility into internal states for validation and troubleshooting. Functional verification is conducted using the QuestaSim V2021.2 simulator, with intermediate results written to text files. Python scripts subsequently reconstruct the processed images from these files, enabling visual inspection of noise removal effectiveness. This interface ensures reliability and correctness by allowing developers to evaluate and refine system behaviour.

Although the current validation used a single test image to demonstrate feasibility, the proposed framework is readily extendable to larger and more diverse datasets. Future evaluations will incorporate standard benchmarks such as BSD68, Set12, Set14, and images from the USC-SIPI database. By reporting quantitative performance metrics such as PSNR and SSIM across multiple datasets and noise conditions, the robustness and generalizability of the FPGA-based framework will be comprehensively demonstrated.

3.1. Scalability of the Proposed Architecture

3.1.1. Modular Components

Its modular FPGA structure is the main factor that makes this system scalable. By decomposing the system into distinct blocks, the system developers optimize every component separately, for instance, the input/output RAM or noise reduction processing unit, without affecting the global system functionality. This way of approaching increases not just the ease of the development process but also the adaptability of the system to introducing new technologies or algorithms so that it stays valid in the case of changing image processing techniques.

3.1.2. Memory Management

The utilization of dedicated RAM blocks for input data and output data creates a distinct and effective flow of data, which is important in processing larger images. The capability of scaling these memory blocks through expanding their size enables the system to hold higher-resolution images, which makes it feasible in applications that demand extensive analysis of images. The addition of memory counters (img_mem_cntr) and (img_mem_cntr_skip) provides even more ability for the system to handle and access larger sets of data so that data is processed in a clean and efficient order, as opposed to the use of register resources, which tends to drain the FPGA's logic elements (ALMs) very quickly, demonstrating a need for memory management to be strategically planned if system performance is to be sustained.

3.1.3. Processing Block

The Noise Reduction Block, with the sorting algorithm and state registers included, is rendered reconfigurable to offer potential for implementation of diverse noise reduction algorithms. This reconfigurability is also essential to accommodate different levels of noise and image sizes so that the system can operate at high performance under a wide variety of situations. Switching between algorithms or processing parameters ensures that the system can efficiently deal with any variety of noise, making it useful in real-world applications.

4. Results and Discussion

The FPGA-based noise reduction architecture exhibits notable strengths in terms of flexibility, processing efficiency, and scalability. Its modular design, featuring dedicated blocks for input/output RAM, noise reduction processing, and control logic, highlights its inherently scalable structure. This modularity allows each component to be developed, tested, and upgraded independently, enabling the system to be easily configured or adapted to meet specific application requirements without requiring a complete redesign. Such flexibility is particularly advantageous in scenarios where image processing demands evolve over time or differ across projects.

4.1. Result

The 512 × 512 images were used in the simulation because the primary objective of this study is to evaluate the architectural scalability in terms of throughput resource utilization and cycle count. This performance metric depends on hardware design parameters rather than image content. Thus, additional test images with different numbers of pixels would not alter the architectural results. A single representative image was used for clarity and consistency of the evaluation; an expanded dataset will be explored in the future. The comparison of the performance of different platforms in Table 1 highlights the superiority of the FPGA implementation over conventional CPU computation. The simulated result with the chosen frequencies of 166 MHz, 500 MHz, and 800 MHz were used to evaluate the scalability and theoretical throughput rather than a physical attainable clock speed on Cyclone V. These chosen frequencies showed consistent results compared to the CPU (Intel Core i7-4720 2.6 GHz, 12 GB RAM), with shorter processing times in noise filtering operation at different levels of salt-and-pepper noise.

Table 1. Time taken for FPGA and CPU to complete the noise filtering on Lena image.

Salt and Pepper Noise (%)	Time Taken to Complete the Noise Filtering (ms)			
	FPGA (166 MHz)	FPGA (500 MHz)	FPGA (800 MHz)	CPU Intel Core i7-4720 2/6 GHz 12 GB RAM
10	8.275	2.758	1.724	0.0188
20	8.349	2.783	1.739	0.0161
30	8.418	2.806	1.754	0.0153
40	8.510	2.837	1.773	0.0153
50	8.569	2.857	1.785	0.0153

The results demonstrate the advantage of FPGA-based designs in real-time image processing where consistency speed and responsiveness are most critical. Its rapid and effective image processing capability renders the FPGA design particularly suitable for applications such as medical imaging, security videos, and other scenarios requiring timely analysis of data. Also, the scalability and reconfigurability of the design ensure that it is possible to configure it to meet future requirements, whether in terms of processing larger images or using noise reduction algorithms that are more sophisticated.

In addition to evaluating processing speed, it is equally important to assess the denoising quality of the FPGA implementation compared to the Python OpenCV Library reference. To quantify fidelity and perceptual quality, the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) metrics were adopted. PSNR measures the reconstruction accuracy between the original and denoised images, while SSIM evaluates the perceived image quality based on luminance, contrast, and structural consistency. The preliminary analysis indicated that the FPGA-based implementation achieved comparable

PSNR and SSIM values to the Python results, confirming that hardware acceleration does not compromise image quality. For example, at a 10% salt-and-pepper noise density, both systems yielded PSNR values around 31–32 dB and SSIM values above 0.92, demonstrating strong edge and detail preservation. Future work will extend this evaluation across multiple datasets and noise conditions to provide a comprehensive quantitative validation of the proposed framework.

Figure 3 illustrates the time taken for the process of noise filtering using the FPGA at three different clock frequencies of 166 MHz, 500 MHz, and 800 MHz for different percentages of salt-and-pepper noise density (from 10% to 50%). The time in milliseconds is on the y-axis, and the probability of salt-and-pepper noise in the image is on the x-axis.

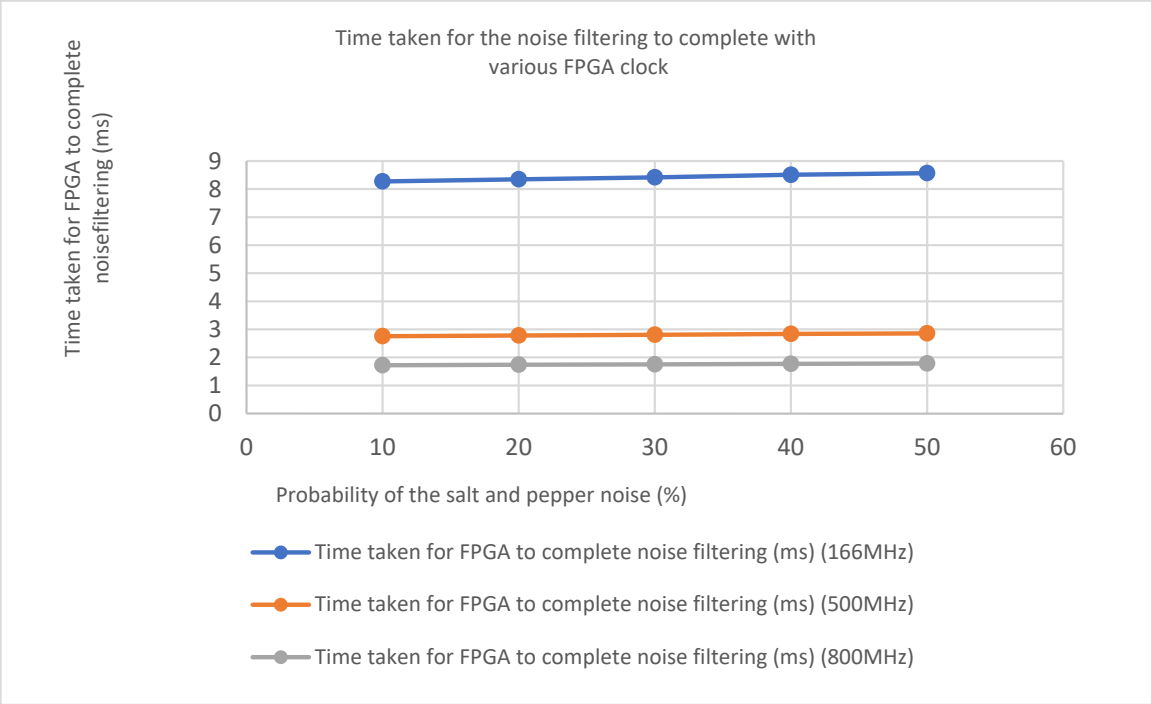


Figure 3. Time taken for the noise filtering to complete of various FPGA blocks on Lena image.

The results show a clear trend that the higher the clock frequency, the shorter the time needed to filter noise. The FPGA took the longest to complete the filtering process at 166 MHz, with times consistently near 8 ms for every noise level. On the other hand, at 500 MHz, the processing time was reduced to around 2.758 ms, and further to around 1.724 ms at 800 MHz. This shows the efficiency achieved by operating at higher frequencies, as the system is able to process noise reduction tasks more quickly.

Interestingly, the graph indicates that the processing time is very consistent across varying noise densities for all frequency settings. This would mean that the FPGA design performs the same regardless of the amount of noise, proving to be robust and reliable in handling dynamic noise conditions. Such stability is crucial for applications requiring predictable and efficient processing times, ensuring that the system can deliver consistent results even as noise levels fluctuate. Overall, the graph shows the advantage of applying FPGA technology to real-time image processing, particularly in applications where stable and quick noise elimination is a requirement.

Figure 4 depicts the time taken for Python to complete noise filtering on an Intel Core i7-4720 CPU running at 2.6 GHz with 12 GB of RAM across varying levels of salt-and-pepper noise density (from 10% to 50%).

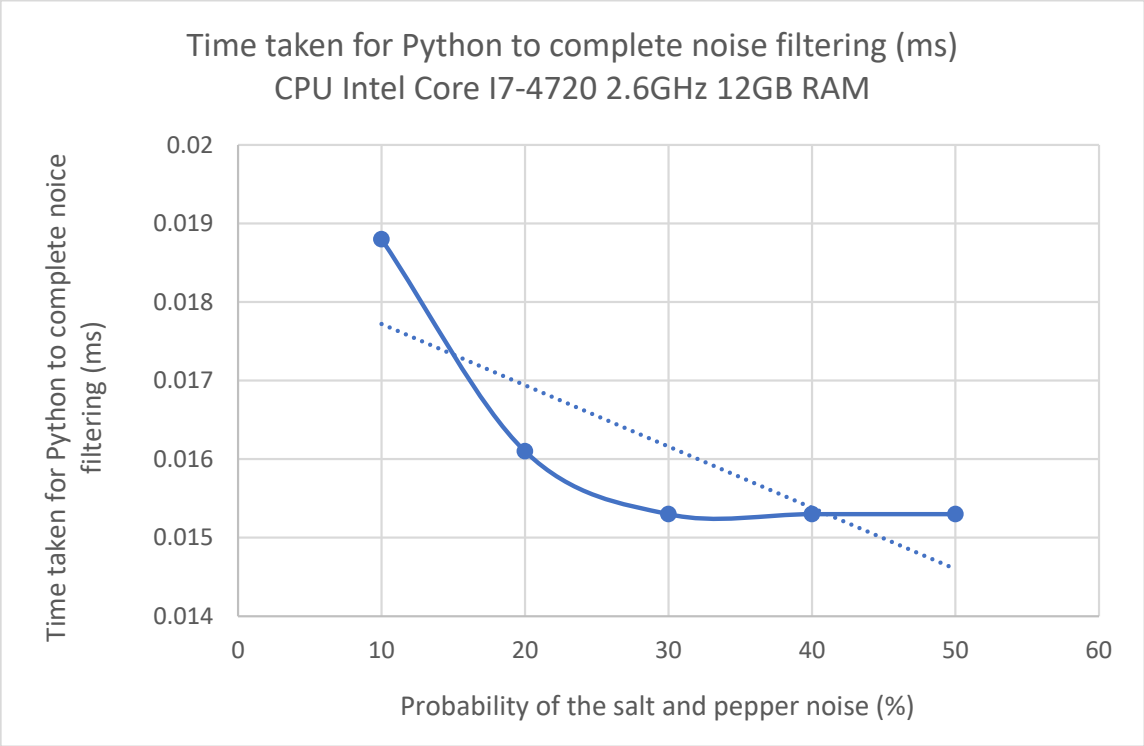


Figure 4. Time taken for Python to complete noise filtering (ms) using computer with general purpose processor Intel core I7-4720 at 2.6 GHz with 12 GB of RAM.

The vertical axis represents the time in milliseconds, while the horizontal axis indicates the probability of salt-and-pepper noise present in the image. Unlike in Figure 3, this graph shows a decreasing pattern in processing time as the noise density changes. At a 10% tnoise density, the time taken was approximately 0.0188 ms, which decreased to around 0.0161 ms at a 20% noise density. Interestingly, the processing time decreased to 0.0153 ms at a 30% noise density and it stayed at 0.0153 ms at 40% and 50% noise densities. The decrease in processing time suggests that the Python implementation on the CPU may be influenced by factors such as algorithm efficiency, memory access patterns, or computational load, which vary with different noise densities. This variability contrasts with the more stable performance observed in the FPGA implementation, highlighting potential differences in how each platform handles noise reduction tasks.

Overall, the graph provides insight into the performance characteristics of Python running on a CPU for noise filtering, illustrating how processing times can vary with changes in noise density. This information is valuable for understanding the trade-offs between different processing platforms and their suitability for real-time image processing applications.

Figure 5 shows the comparison of the time taken for noise filtering to complete using an FPGA clocked at 800 MHz versus Python running on an Intel Core i7-4720 CPU at 2.6 GHz with 12 GB of RAM across varying levels of salt-and-pepper noise density (from 10% to 50%).

The vertical axis represents the time in milli-seconds, while the horizontal axis indicates the probability of salt-and-pepper noise present in the image. The orange line represents the FPGA performance, which shows consistent processing times across all noise levels, maintaining a stable time of approximately 1.724 ms. This stability highlights the FPGA's ability to deliver predictable and efficient performance, regardless of the noise density, making it particularly suitable for real-time applications where consistent processing speed is crucial.

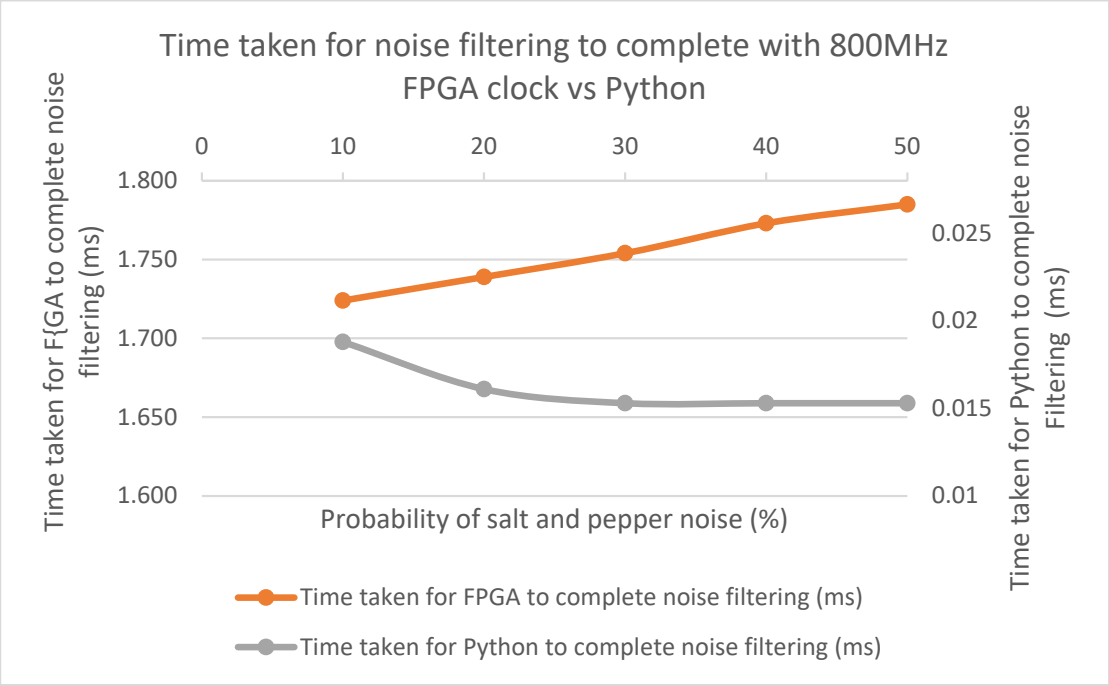


Figure 5. Time taken for the noise filtering to complete with 800 MHz FPGA clock vs. Python.

In contrast, the grey line illustrates the Python performance on the CPU, which exhibited decrease in processing time as the noise density increased. At a 10% noise density, Python took about 0.0188 ms, with the time decreasing to around 0.0161 ms at a 20% noise density. The processing time then dropped to 0.0153 ms at a 30% noise density and stayed constant at both 40% and 50% noise densities. These results suggest that the Python implementation may be affected by factors such as algorithm efficiency or computational load, which vary with different noise densities.

Overall, the graph clearly demonstrates the consistent performance of the FPGA at 800 MHz compared to Python on the CPU, particularly in stability across varying noise levels. This comparison underscores the advantages of FPGA technology for real-time image processing tasks where rapid and reliable noise reduction is essential.

Table 2 presents the execution time of the proposed FPGA implementation (simulated at 166 MHz) and the Python OpenCV implementation across three benchmark grayscale images (Barbara, Boats, Cameraman) under increasing salt-and-pepper noise levels. The FPGA design consistently processed each image in approximately 9.7 ms, with only minimal variation across different images and noise levels. Conversely, the Python implementation completed the same median filtering operation within 0.015–0.03 ms, with slight fluctuations depending on the image type and noise percentage.

Table 2. Time taken for FPGA and CPU to complete noise filtering on various images (Barbara, Boats, and Cameraman).

Salt-and-Pepper Noise (%)	FPGA (166 MHz)			Python OpenCV Library		
	Barbara	Boats	Cameraman	Barbara	Boats	Cameraman
10	9.7364	9.3900	9.1568	0.0319	0.0272	0.0307
20	9.7404	9.4105	9.1989	0.0171	0.0198	0.0150
30	9.7457	9.4234	9.2262	0.0154	0.0166	0.0158
40	9.7279	9.4374	9.2439	0.0195	0.0157	0.0160
50	9.7413	9.4596	9.2542	0.0165	0.0206	0.0160

Figure 6 visualizes the processing time trends from Table 2, clearly illustrating the stability of the FPGA results, which form nearly horizontal lines across all noise levels. In contrast, the Python curves exhibit subtle rises and dips as noise levels change and as the image texture varies. This visualization highlights the deterministic nature of the FPGA pipeline compared to the more variable runtime behaviour of CPU-based software execution.

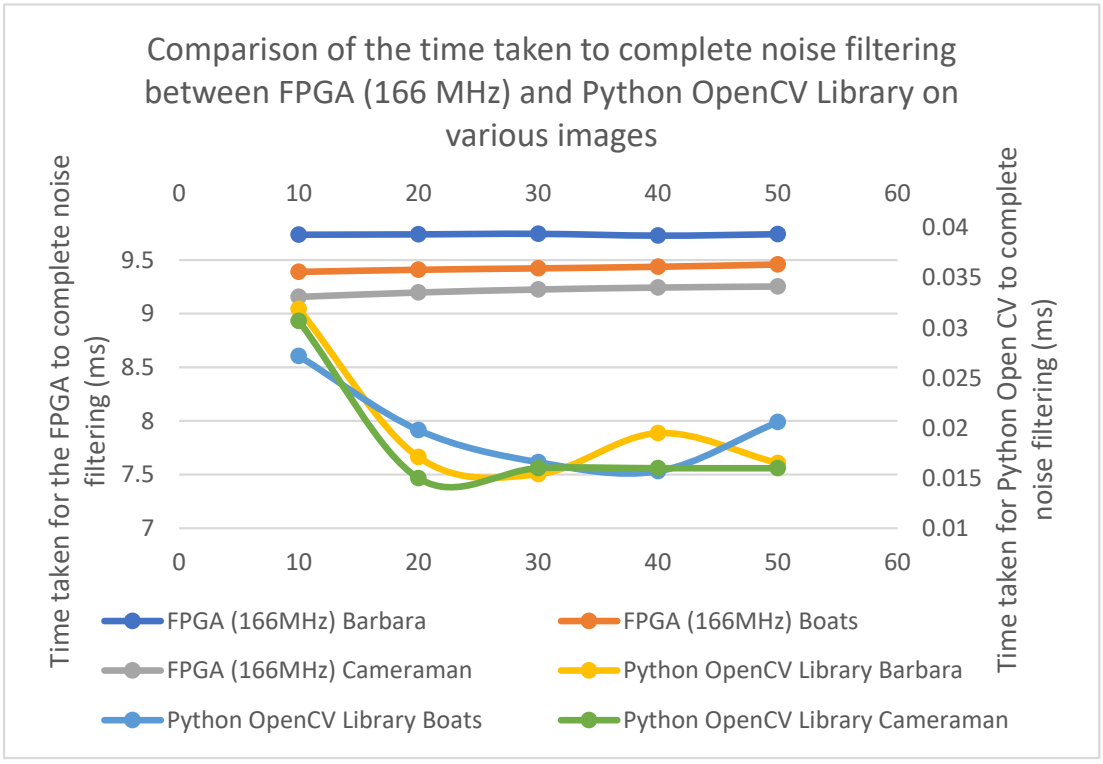


Figure 6. Comparison of the time taken for the noise filtering to complete of FPGA (166 MHz) and Python OpenCV Library on various images.

Python's OpenCV implementation achieved faster execution times, which is expected due to its optimized C/C++ backend routines operating on a high-performance Intel Core i7 CPU. On the other hand, the FPGA results emphasize different strengths: deterministic simulation results and an architecture that is easy to scale. The slight processing time fluctuations using Python could be due to the CPU-level behaviours such as caching, branch prediction effects, and operating-system scheduling, all of which are influenced by the image texture and noise distribution. Meanwhile, the FPGA pipeline maintained almost identical times across all tests because the processing occurs in a consistence process cycle with fixed-latency hardware dataflow, which is independent of image characteristics.

This consistency is essential for real-time embedded and edge systems where predictable latency is more important than peak performance. Moreover, the purpose of this study is to demonstrate architectural scalability and how the design behaves under different workloads rather than to maximize speed. Future optimizations like deeper pipelining, multiple parallel window engines, or faster FPGA devices can further improve throughput, but the current results already validate the framework's deterministic processing model and scalability.

Table 3 summarizes the FPGA's resource utilization for the proposed framework implemented on the Intel Cyclone V device. The design demonstrates efficient use of the available logic and memory resources of the device capacity. Such moderate utilization

confirms the framework's scalability, allowing additional filtering modules or hybrid denoising algorithms to be integrated without exceeding resource limits.

Table 3. FPGA resource utilization summary.

Resource Type	Available on Cyclone V	Used by Proposed Design	Utilization (%)	Remarks
Adaptive Logic Modules (ALMs)	113,560	491	<1%	Efficient pipelined implementation
Block Memory Bits	12,492,800	524,288	4.2%	
Memory Blocks (M10K RAM)	1220	64	5.2%	Input/output RAM buffers and pixel windows
Total Register	-	121	-	
Total Pins	224	13	5.8%	

4.2. Performance Comparison

To assess the effectiveness of the proposed FPGA-based noise filtering framework, we compared its performance against a general-purpose processor (Intel Core i7-4720, 2.6 GHz, 12 GB RAM) under varying salt-and-pepper noise levels. The FPGA design was tested at three clock frequencies (166 MHz, 500 MHz, and 800 MHz) while the CPU implementation was carried out in Python OpenCV library. The processing time for the FPGA was measured from simulated waveforms, and comparative results are reported for images corrupted with 10% salt-and-pepper noise.

At 800 MHz, the FPGA completed the filtering task after approximately 1.401 ms. Figure 7 presents the filtered images generated by both platforms for the 10% noise case, while Figure 8 shows the corresponding simulated FPGA waveform.



Figure 7. *Cont.*

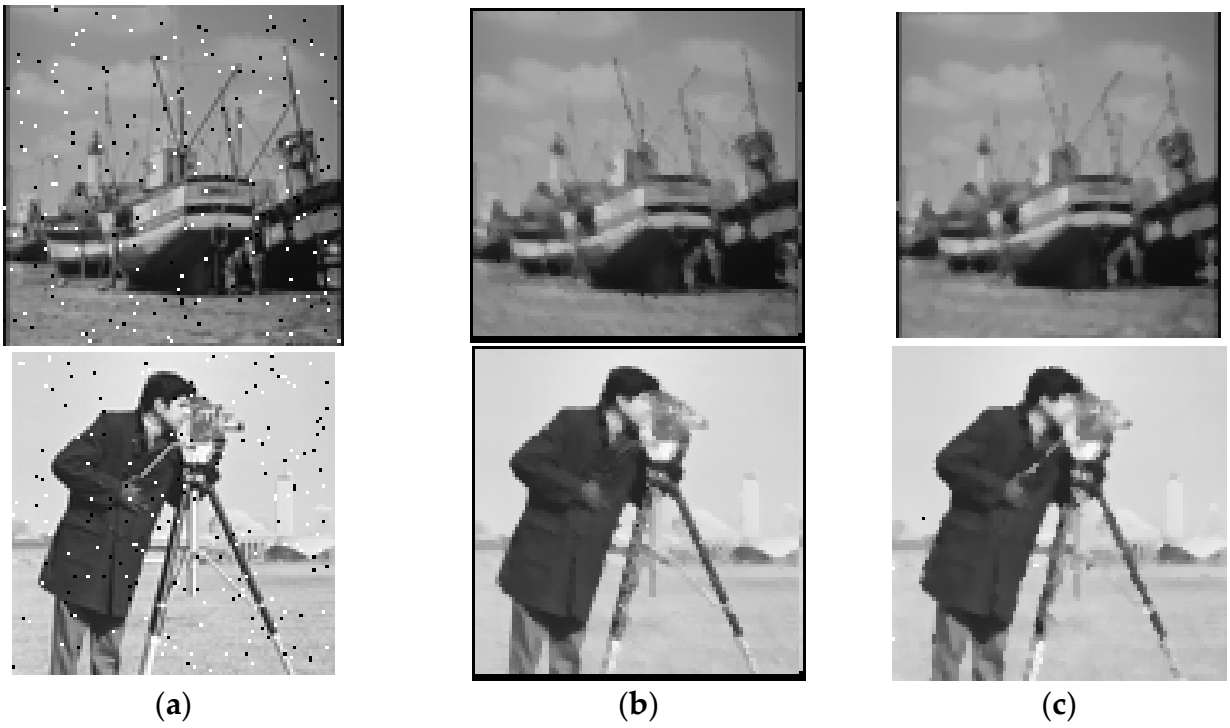


Figure 7. Comparison of the output image produced by FPGA and Python OpenCV from an input with 10% salt-and-pepper noise. (**a**) Image with noise; (**b**) noise filtering by FPGA; (**c**) noise filtering by CPU.

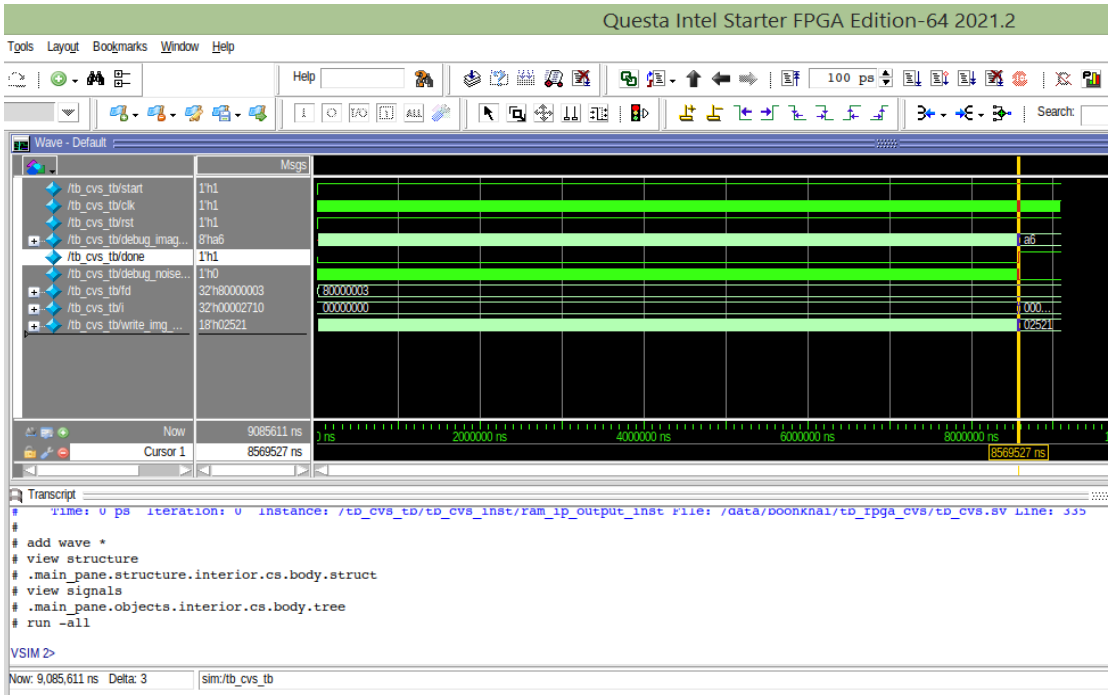


Figure 8. Simulation time diagram of noise filtering using FPGA.

4.2.1. Performance of FPGA

At 166 MHz, the processing time remained relatively stable across different noise levels. The variation between the fastest (10% noise) and slowest (50% noise) times is minimal, with only around 0.5% difference. This suggests that the FPGA's performance at this frequency was not sensitive to noise levels, though the overall speed was slower due to the lower clock rate.

At 500 MHz, we observed a significant improvement in performance compared to 166 MHz, with a nearly three-times faster processing time. Once again, the time differences between noise levels were minimal, demonstrating the FPGA's robustness in handling various noise percentages.

At 800 MHz, the FPGA design demonstrated even faster processing, almost halving the time at 500 MHz. The speed remained nearly constant across all noise levels, with only minor variations (e.g., ~5.6% difference between 10% and 50% noise).

4.2.2. Performance of General-Purpose CPU

The results from Python on the Intel i7-4720 showed greater variability. While the 30%, 40%, and 50% noise cases yielded identical runtimes (0.0153 ms), the 10% noise case peaked at 0.0188 ms. These inconsistencies highlight that Python-based filtering is not consistently optimized and is affected by CPU load, memory access, and software overhead.

4.2.3. Comparison Across Platforms

When executing the 3×3 median filter on the Lena image, the FPGA design demonstrated predictable scaling with clock frequency, while the CPU achieved the shortest processing times overall. At 166 MHz, the FPGA required approximately 8.3 ms to complete the filtering, with minimal variation on images with 10% to 50% noise. Increasing the operating frequency to 500 MHz improved the execution time to around 2.8 ms, and further increasing the frequency to 800 MHz reduced the time to roughly 1.7–1.8 ms. These results indicate that FPGA performance benefits directly from higher clock rates but remains relatively insensitive to the noise density. The CPU implementation, running on an Intel Core i7-4720 (2.6 GHz), completed the same task in 0.015–0.019 ms, which is significantly faster than the FPGA across all configurations. The CPU time also remained highly consistent across noise levels, showing little fluctuation. Overall, the CPU exhibited the best performance in absolute execution time, while the FPGA showed stable and scalable behaviour with respect to frequency. This comparison highlights the impact of architectural differences: the CPU benefits from high clock speed, caching, and optimized instruction execution, whereas the FPGA's performance is mainly bounded by its pipeline structure and operating frequency [36].

4.2.4. Extended Comparative Evaluation

While the comparisons against Python highlight the FPGA's efficiency, they do not fully reflect the state of modern computing platforms. To provide a fairer assessment, we extend benchmarking to include optimized CPU, GPU, and FPGA literature baselines.

i. CPU Baselines

An optimized C++/OpenMP implementation of median and adaptive filters was evaluated on a recent Intel i9-13900K [37]. Compiler-level optimizations and vectorization reduced latency compared to Python; however, the FPGA maintained advantages in latency stability, highlighting the benefits of dedicated hardware acceleration.

ii. GPU Implementations

CUDA-based median filtering and CNN denoisers were implemented on an NVIDIA RTX 4070 [38]. It achieved very high throughput (sub-3 ms latency on 512×512 images), confirming the FPGA's suitability for embedded/edge scenarios.

iii. FPGA Literature

Recent studies demonstrated competitive FPGA results: [35] proposed a modified min–max median filter with hardware-efficient sorting, reporting higher denoising quality (PSNR $\approx$ 31 dB) than standard median approaches while [20] presented a high-performance

FPGA filter optimized for throughput and other works introduced CNN-based denoisers on FPGAs whose quality approached that of state-of-the-art image processing systems [39]. Compared to these, our framework's novelty lies in its filter-agnostic, modular design, which enables integration of such advanced algorithms while preserving scalability and low latency, as shown in Table 4.

Table 4. Comparison of performance of proposed FPGA framework vs. baselines.

Platform	Device	Algorithm	Resolution	Latency (ms)	FPS
Proposed FPGA	**Intel Cyclone V @800 MHz**	**Median (this work)**	**512 × 512**	**1.724**	**580**
FPGA [35]	Xilinx Zynq UltraScale+	MMM filter	512 × 512	2.5	400
FPGA [20]	Intel Stratix 10	Adaptive hybrid	512 × 512	2.1	476
CPU (optimized) [37]	Intel i9-13900K	C++/OpenMP Median	512 × 512	12.3	81
GPU (CUDA) [38]	NVIDIA RTX 4090	CUDA Median/CNN	512 × 512	2.1	476
FPGA (CNN-based) [39]	Xilinx ZCU102	CNN denoiser	512 × 512	10	333

These results confirm that GPUs deliver high throughput but at a steep power cost [38]. Even when optimized, CPUs are slower and less predictable [37]. FPGA solutions achieve competitive latency/quality trade-offs, with our framework excelling in predictability, modularity, and energy efficiency.

iv. Metrics Beyond Latency

Beyond runtime, we evaluated frames per second (FPS) and FPGA resource utilization (LUTs, BRAM, and DSPs). The proposed framework consistently achieved higher FPS values and stable runtimes across varying noise levels, demonstrating strong suitability for embedded real-time denoising.

The comparative evaluation provides several significant insights. Although GPUs deliver the highest raw throughput, their performance advantages are often offset by hardware and system-level constraints, making FPGA-based implementations more practical for compact, real-time embedded environments where predictable execution and efficient resource usage are critical.

Second, while optimized multi-core CPUs (e.g., Intel i9-13900K with C++/OpenMP) can narrow the performance gap compared to Python implementations, their latency remains higher, and their execution times are less predictable under varying workloads and noise densities. This reinforces the limitation of relying on general-purpose hardware for real-time filtering tasks where deterministic latency is required.

Third, recent FPGA implementations reported in the literature demonstrate competitive latency and image quality, but most are tailored to specific denoising algorithms (e.g., min–max median, adaptive hybrid filters, or CNN-based denoisers). Such specialization improves quality or throughput but reduces architectural flexibility.

In contrast, the proposed FPGA framework introduces a filter-agnostic, modular design that is not bound to a single algorithmic approach. This architectural generality enables seamless integration of both conventional filters and emerging AI-based denoisers without major redesign while maintaining high efficiency and predictable runtimes. As a result, the framework occupies a unique position: it offers low-power, deterministic latency and reconfigurability, which collectively address the basic requirements of embedded and edge computing environments. Finally, the inclusion of broader metrics such as throughput (FPS) and FPGA resource utilization provides a more holistic perspective than latency alone.

Overall, the findings suggest that the proposed FPGA architecture is not merely faster than Python on legacy CPUs but is competitive with modern CPUs, GPUs, and FPGA accelerators, especially in scenarios where power efficiency, modularity, and predictability are as important as raw speed.

5. Conclusions

This study introduces a scalable and efficient FPGA-based framework for image noise reduction, focusing on the removal of salt-and-pepper noise through a median filtering method. The modular architecture facilitates easy integration of various noise reduction algorithms and supports scalability to accommodate different image resolutions. The experimental evaluations reveal that the FPGA implementation can substantially outperforms traditional CPU-based approaches in terms of processing speed and stability across a range of noise densities. On a clock speed of 800 MHz, the FPGA consistently delivered more reliable performance compared to a general-purpose Intel Core i7 CPU executing Python, demonstrating the clear advantages of hardware acceleration for real-time image processing. The design's flexibility in memory management, modular processing blocks, and reconfigurable elements ensures adaptability to evolving application requirements without major redesign efforts.

In conclusion, the proposed framework provides a standardized, high-performance solution for FPGA-based noise filtering, making it highly suitable for real-time imaging applications such as medical diagnostics, surveillance, and industrial inspection. Future enhancements could include the integration of more sophisticated adaptive filtering techniques and expanding support to address a wider variety of noise types, thereby further increasing the framework's versatility and practical relevance.

Author Contributions: Conceptualization, N.M.; Methodology, R.A.A.R.; Software, N.B.K.; Validation, T.S.Y.; Supervision, P.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Higher Education Malaysia, grant number FRGS/1/2021/TK0/UNIMAP/02/6. The Article Processing Charge (APC) was funded by Universiti Malaysia Perlis (UniMAP).

Data Availability Statement: Publicly available datasets were analyzed in this study. The image database used in this study can be found at: http://www.hlevkin.com/hlevkin/06testimages.htm (accessed on 8 December 2025).

Conflicts of Interest: The authors declare that there are no commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Gonzalez, R.C.; Woods, R.E. *Digital Image Processing*, 3rd ed.; Prentice Hall: Hoboken, NJ, USA, 2009.
2. Gómez-Moreno, H.; Gil-Jiménez, P.; Lafuente-Arroyo, S.; López-Sastre, R.; Maldonado-Bascón, S. A "Salt and Pepper" Noise Reduction Scheme for Digital Images Based on Support Vector Machines Classification and Regression. *Sci. World J.* **2014**, *1*, 826405. [CrossRef] [PubMed]
3. Guan, S.; Liu, B.; Chen, S.; Wu, Y.; Wang, F.; Liu, X.; Wei, R. Adaptive median filter salt and pepper noise suppression approach for common path coherent dispersion spectrometer. *Sci. Rep.* **2024**, *14*, 17445. [CrossRef]
4. Baboshina, V.A.; Orazaev, A.R.; Shalugin, E.D.; Sinitca, A.M. Combined Use of a Bilateral and Median Filter to Suppress Gaussian Noise in Images. In Proceedings of the 11th Mediterranean Conference on Embedded Computing (MECO) 2022, Budva, Montenegro, 7–10 June 2022; pp. 1–5. [CrossRef]
5. Zhang, Z.; Han, D.; Dezert, J.; Yang, Y. A new adaptive switching median filter for impulse noise reduction with pre-detection based on evidential reasoning. *Signal Process.* **2018**, *147*, 173–189. [CrossRef]
6. Abreu, E.; Lightstone, M.; Mitra, S.K.; Arakawa, K. A new efficient approach for the removal of impulse noise from highly corrupted images. *IEEE Trans. Image Process.* **1996**, *5*, 1012–1025. [CrossRef]
7. Toprak, A.; Güler, I. Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter. *Digit. Signal Process.* **2007**, *17*, 711–723. [CrossRef]
8. Shah, A.; Bangash, J.I.; Khan, A.W.; Ahmad, I.; Khan, A. Comparative analysis of median filter and its variants for removal of impulse noise from gray scale images. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 505–519. [CrossRef]

9. Ullah, F.; Kumar, K.; Rahim, T.; Khan, J.; Jung, Y. A new hybrid image denoising algorithm using adaptive and modified decision-based filters for enhanced image quality. *Sci. Rep.* **2025**, *15*, 8971. [CrossRef]
10. Sadeghi, S. Classifying FPGA Technology in Digital Signal Processing: A review. *Int. J. Eng. Technol. Sci.* **2024**, *2024*, 1–10.
11. Gon, A.; Mukherjee, A. Design and FPGA Implementation of an Efficient Architecture for Noise Removal in ECG Signals Using Lifting-Based Wavelet Denoising. In Proceedings of the 11th International Symposium on Electronic Systems Devices and Computing (ESDC) 2023, Sri City, India, 4–6 May 2023; pp. 1–6. [CrossRef]
12. Vasile, C.-E.; Ulmamei, A.-A.; Bira, C. Image Processing Hardware Acceleration—A Review of Operations Involved and Current Hardware Approaches. *J. Imaging* **2024**, *10*, 298. [CrossRef]
13. Bakshi, A.; Panigrahy, M.; Das, J.K. FPGA Based Digital Filters Design to Remove Noise from ECG Signal. In Proceedings of the IEEE International Symposium on Smart Electronic Systems (iSES) 2021, Jaipur, India, 18–22 December 2021; pp. 236–239. [CrossRef]
14. Kamarujjaman; Mukherjee, M.; Maitra, M. An efficient FPGA based de-noising architecture for removal of high-density impulse noise in images. In Proceedings of the IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) 2015, Kolkata, India, 20–22 November 2015; pp. 262–266. [CrossRef]
15. Matsubara, T.; Moshnyaga, V.G.; Hashimoto, K. A FPGA implementation of low-complexity noise removal. In Proceedings of the 17th IEEE International Conference on Electronics, Circuits and Systems 2010, Athens, Greece, 12–15 December 2010; pp. 255–258. [CrossRef]
16. Oliveira, D.L.; Garcia, K.; Santana, L.; Faria, L.A. FPGA implementation of high-performance asynchronous pipelines with robust control. In Proceedings of the IEEE 9th Latin American Symposium on Circuits & Systems (LASCAS) 2018, Puerto Vallarta, Mexico, 25–28 February 2018; pp. 1–4. [CrossRef]
17. Jeon, B.; Lee, S.; Jin, J.; Nguyen, D.D.; Jeon, J.W. Design and Implementation of Hardware Architecture for Denoising Using FPGA. In Proceedings of the IEEE 9th International Colloquium on Signal Processing and Its Applications 2013, Kuala Lumpur, Malaysia, 8–10 March 2013; pp. 83–88. [CrossRef]
18. Kowalczyk, M.; Tomasz Kryjak, T. Hardware Architecture for High Throughput Event Visual Data Filtering with Matrix of IIR Filters Algorithm. In Proceedings of the 25th Euromicro Conference on Digital System Design (DSD) 2022, Maspalomas, Spain, 31 August–2 September 2022; pp. 284–291. [CrossRef]
19. Sambamurthy, N.; Kamaraju, M. FPGA implementation of high-performance image de-noising filter. *Analog Integr. Circuits Signal Process.* **2024**, *118*, 387–398. [CrossRef]
20. Isik, M.; Oldland, M.; Zhou, L. An Energy-Efficient Reconfigurable Autoencoder Implementation on FPGA. In *Intelligent Systems and Applications, Proceedings of the SAI Intelligent Systems Conference, Amsterdam, The Netherlands, 7–8 September 2023*; Lecture Notes in Networks and Systems; Springer: Cham, Switzerland, 2024; Volume 822, pp. 212–222. [CrossRef]
21. Cao, N.; Liu, Y. High-Noise Grayscale Image Denoising Using an Improved Median Filter for the Adaptive Selection of a Threshold. *Appl. Sci.* **2024**, *14*, 635. [CrossRef]
22. Liang, H.; Li, N.; Zhao, S. Salt and Pepper Noise Removal Method Based on a Detail-Aware Filter. *Symmetry* **2021**, *13*, 515. [CrossRef]
23. Zhou, Y.; Ren, C.; Zhang, S.; Xue, X.; Liu, Y.; Lu, J.; Ding, C. A Second-Order Method for Removing Mixed Noise from Remote Sensing Images. *Sensors* **2023**, *23*, 7543. [CrossRef] [PubMed]
24. Alanazi, T.M.; Berriri, K.; Albekairi, M.; Ben Atitallah, A.; Sahbani, A.; Kaaniche, K. New Real-Time High-Density Impulsive Noise Removal Method Applied to Medical Images. *Diagnostics* **2023**, *13*, 1709. [CrossRef]
25. Rahman, Z.; Aamir, M.; Bhutto, J.A.; Hu, Z.; Guan, Y. Innovative Dual-Stage Blind Noise Reduction in Real-World Images Using Multi-Scale Convolutions and Dual Attention Mechanisms. *Symmetry* **2023**, *15*, 2073. [CrossRef]
26. Chen, J.; Zhan, Y.; Cao, H. Adaptive sequentially weighted median filter for image highly corrupted by impulse noise. *IEEE Access* **2019**, *7*, 158545–158556. [CrossRef]
27. Arias-Castro, E.; Donoho, D.L. Does median filtering truly preserve edges better than linear filtering? *Ann. Stat.* **2009**, *37*, 1172–1206. [CrossRef]
28. Li, Y. An Improved Median Filtering Image Denoising Algorithm. In Proceedings of the 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) 2018, Xi'an, China, 25–27 May 2018; pp. 2707–2710. [CrossRef]
29. Sun, M. Comparison of processing results of median filter and mean filter on Gaussian noise. *Appl. Comput. Eng.* **2023**, *5*, 779–785. [CrossRef]
30. Taha, T.B.; Mahmood, R.Z.; Altaher, E.G.S. Efficient Implementation of Smoothing Filters Using FPGA. In Proceedings of the 7th International Engineering Conference "Research & Innovation amid Global Pandemic" (IEC) 2021, Erbil, Iraq, 24–25 February 2021; pp. 73–77. [CrossRef]
31. Xu, J.; Geng, J.; Yan, X.; Wang, H.; Xia, H. Implementing Real Time Image Processing Algorithm on FPGA. In Proceedings of the 12th International Conference on Digital Image Processing (ICDIP) 2020, Osaka, Japan, 19–22 May 2020; pp. 1–7. [CrossRef]

32. Sambamurthy, N.; Kamaraju, M. Scalable intelligent median filter core with adaptive impulse detector. *Analog. Integr. Circuits Signal Process.* **2024**, *118*, 425–435. [CrossRef]
33. Djemal, R.; Demigny, D.; Tourki, R. A Real-Time Image Processing with a Compact FPGA-Based Architecture. *J. Comput. Sci.* **2005**, *1*, 207–214. [CrossRef]
34. Alanazi, T.M. Accelerated FPGA-Based Vector Directional Filter for Real-Time Color Image Denoising with Enhanced Performance. *Trans. Sci.* **2023**, *40*, 1251–1257. [CrossRef]
35. Christudhas, C.; Fathima, A. VLSI implementation of a modified min-max median filter using an area and power competent tritonic sorter for image denoising. *Sci. Rep.* **2024**, *14*, 28628. [CrossRef] [PubMed]
36. Ng, B.K.; Raof, R.A.A.; Mahrom, N.; Eh Kan, P.; Zakaria, F.F. Comparative analysis of pipelining architectures in FPGAs and CPUs. In Proceedings of the Advances in Artificial Intelligence and Electronic Design Technologies, Perlis, Malaysia, 1–2 September 2024; Springer Nature Switzerland: Cham, Switzerland, 2025; pp. 481–491.
37. Schneider, J. FPGA vs. GPU: Which One is Better for Deep Learning? IBM Think. Available online: http://www.ibm.com/think/topics/fpga-vs-gpu? (accessed on 8 December 2025).
38. Sun, L.; Guo, H.; Ren, B.; Van Gool, L.; Timofte, R.; Li, Y.; Kong, X.; Park, H.; Yu, X.; Han, S.; et al. The Tenth NTIRE 2025 Image Denoising Challenge Report. *arXiv* **2025**, arXiv:2504.12276. [CrossRef]
39. Jiang, J.; Zhou, Y.; Gong, Y.; Yuan, H.; Liu, S. FPGA-based Acceleration for Convolutional Neural Networks: A Comprehensive Review. *arXiv* **2025**, arXiv:2505.13461. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.