Summary of Research Paper

Underwater Image Enhancement Using FPGA-Based Gaussian Filters with Approximation Techniques
Summary Date: August 1, 2025

---

1. **Introduction / Abstract**
   The paper addresses the problem of poor visual quality in underwater images caused by natural phenomena like light absorption and scattering, which create haziness and noise. To improve these images, the authors propose an efficient method using **Gaussian filters** (a mathematical tool for smoothing images and reducing noise) implemented on **FPGA** (Field-Programmable Gate Array, a type of reconfigurable hardware that allows fast and parallel processing). The key innovation is a pipeline architecture combined with approximate adders (simplified arithmetic units that trade some accuracy for better speed and lower power use). The results show over 150% speed improvement and more than 34% power reduction, with some increase in hardware area. This trade-off suits error-tolerant applications like image and video processing.

2. **Methodology**
   The researchers designed a pipeline Gaussian filter architecture on FPGA, which processes image pixels in stages to increase throughput. They used line buffers to store image rows and window buffers to hold local pixel areas for filtering, minimizing memory access. The Gaussian filter applies a 3×3 kernel (a small matrix of weights) to smooth images by convolution (a weighted sum of neighboring pixels). To reduce hardware complexity and power, they introduced approximate adders in the filter's arithmetic units, testing ten different designs that simplify addition with controlled errors. They simulated the filter in MATLAB with real underwater images corrupted by Gaussian noise and evaluated image quality using metrics like PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index). Finally, they synthesized the designs on an Intel MAX10 FPGA to measure power, speed, and area.

3. **Theory / Mathematics**
   The Gaussian filter is based on the two-dimensional Gaussian function:

$$
f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}
$$

where (f(x,y)) is the filter weight at position ((x,y)), and (\sigma) controls the spread of the filter. This function creates a bell-shaped curve emphasizing central pixels and smoothing out noise. The convolution operation applies this kernel over the image pixels:

$$
h[i,j] = \sum_{k=1}^9 W_k P_k
$$

where (W_k) are kernel weights and (P_k) are pixel values in the 3×3 window.

Image quality metrics used include:
- **PSNR**:

$$
PSNR(f,g) = 20 \log_{10} \left(\frac{2^B - 1}{\sqrt{MSE(f,g)}}\right)
$$

measuring signal fidelity, where (B) is bit depth.
- **MSE (Mean Square Error)**:

$$
MSE(f,g) = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (f_{ij} - g_{ij})^2
$$

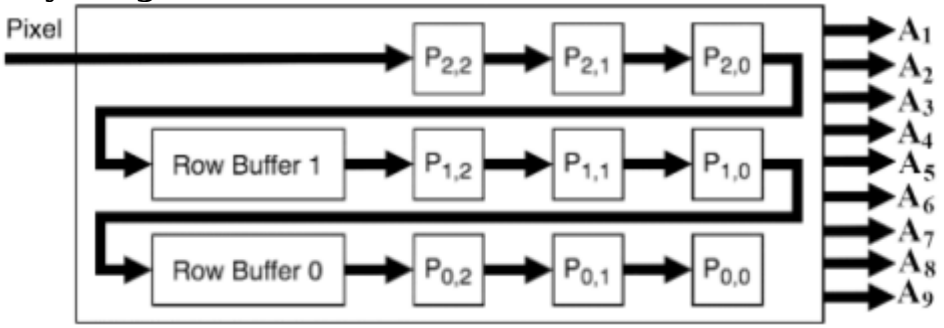
quantifies average pixel difference.
- **SSIM**:

$$
SSIM(f,g) = \frac{(2\mu_f \mu_g + C_1)(2\sigma_{fg} + C_2)}{(\mu_f^2 + \mu_g^2 +
$$

assesses perceptual similarity considering luminance and contrast.

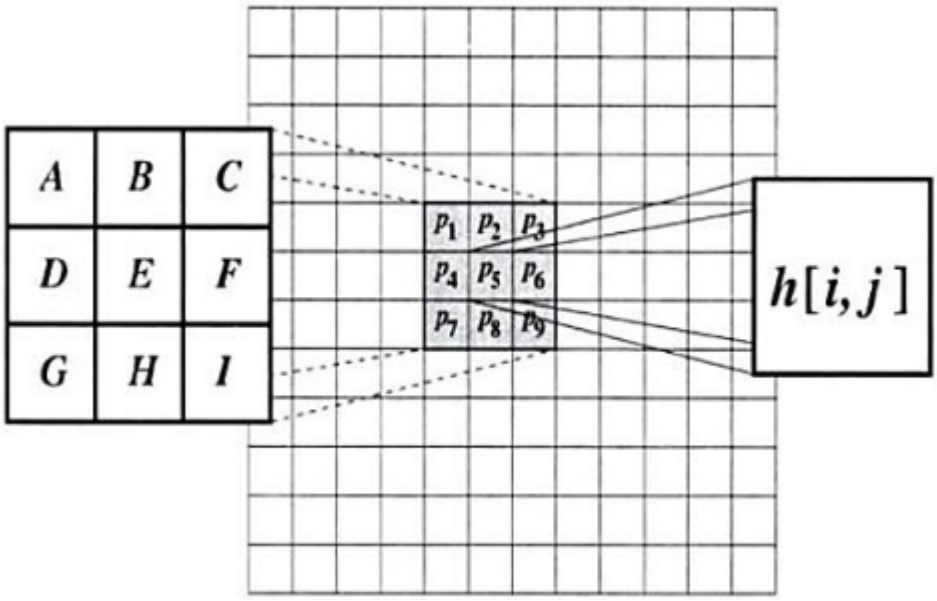
Error Distance (ED) and related metrics measure spatial accuracy of pixel positions.

1. **Key Diagrams or Visual Elements**

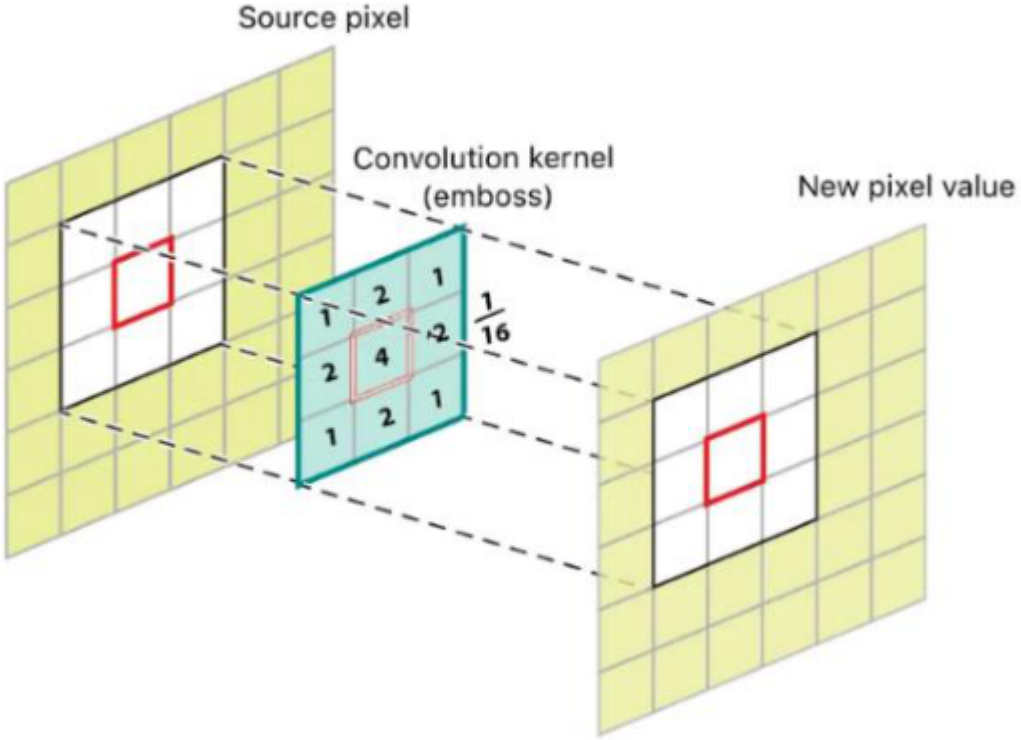


2. **Figure 1:** Shows the delay line buffer structure, which stores recent pixel values to optimize memory access during convolution. This

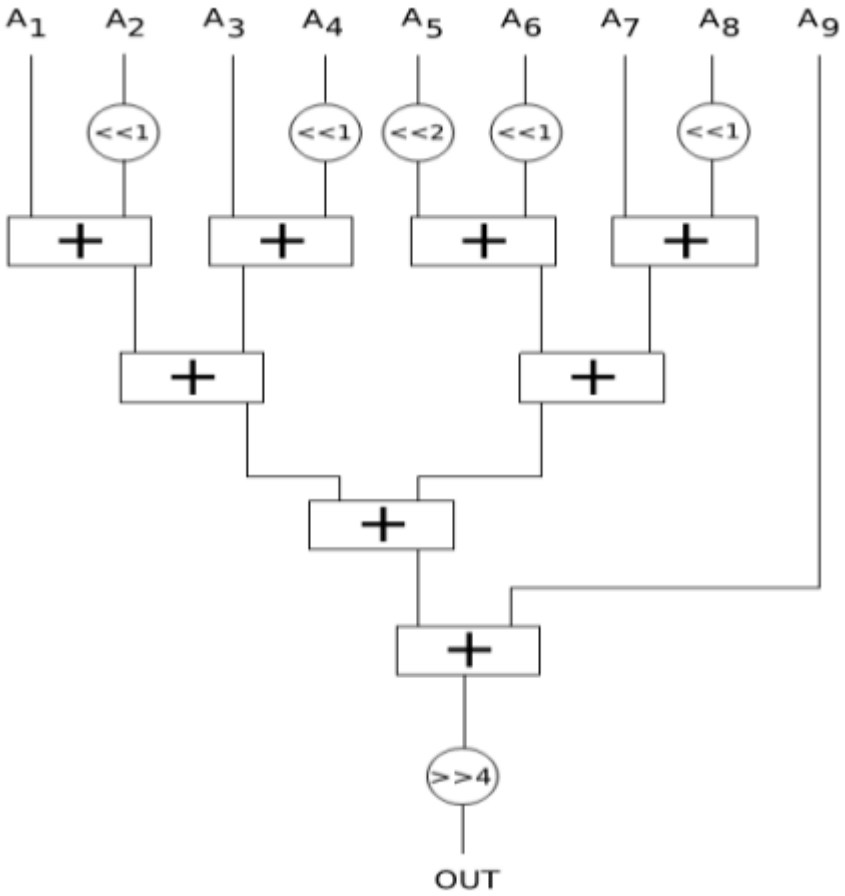
reduces the need to repeatedly access large image memory.



3. **Figure 2:** Illustrates the convolution operation with a 3×3 kernel sliding over the image pixels, showing how each output pixel is computed from weighted sums of neighbors.

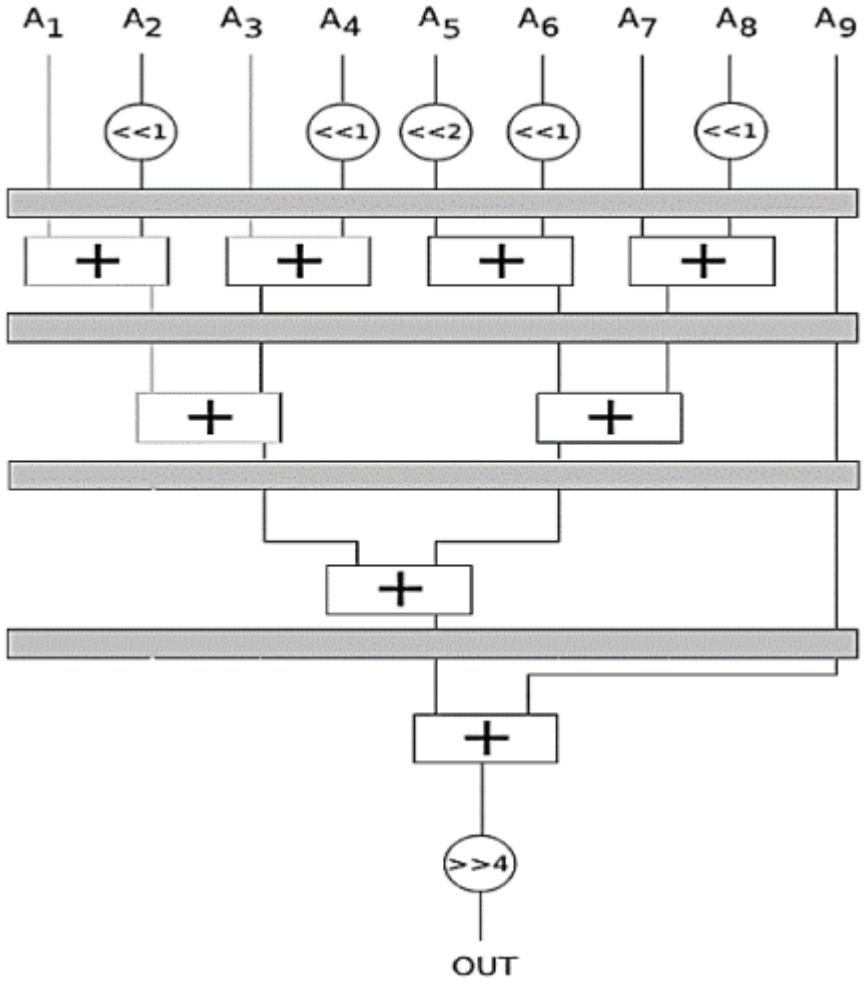


4. **Figure 3:** Displays the Gaussian filter kernel weights used in the 3×3 filter.

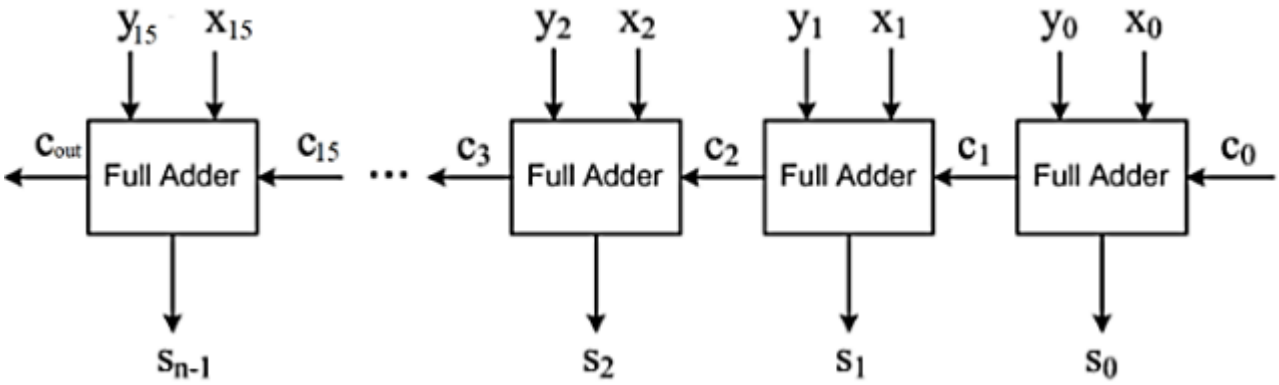


5. **Figure 4:** Block diagram of the Gaussian filter implementation, highlighting the use of adders and shifters for multiplication/division by

powers of two.

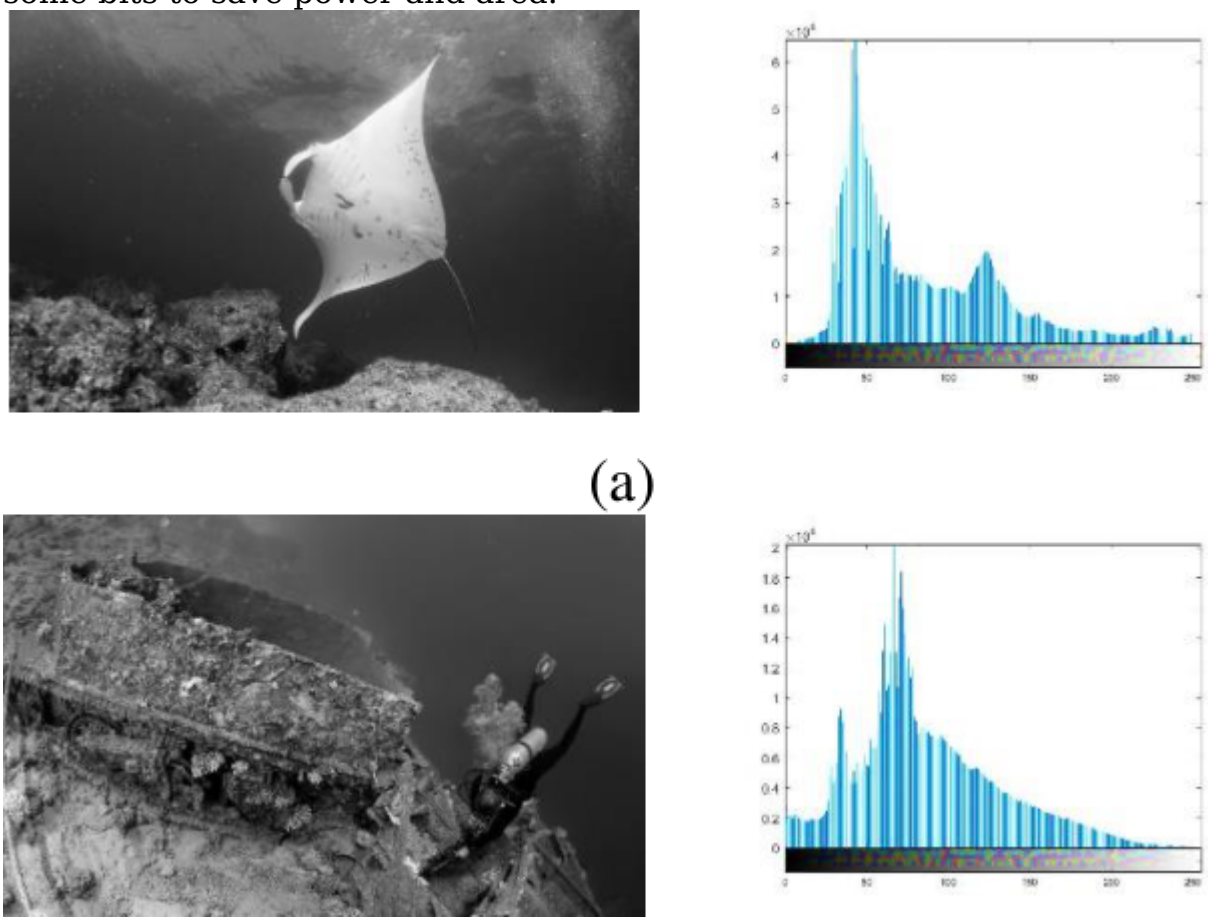


6. **Figure 5:** Depicts the pipeline structure of the Gaussian filter, dividing computation into four stages to increase processing speed.



7. **Figure 6:** Shows the 16-bit carry ripple adder architecture used for addition in the filter, where approximate adders replace precise ones in

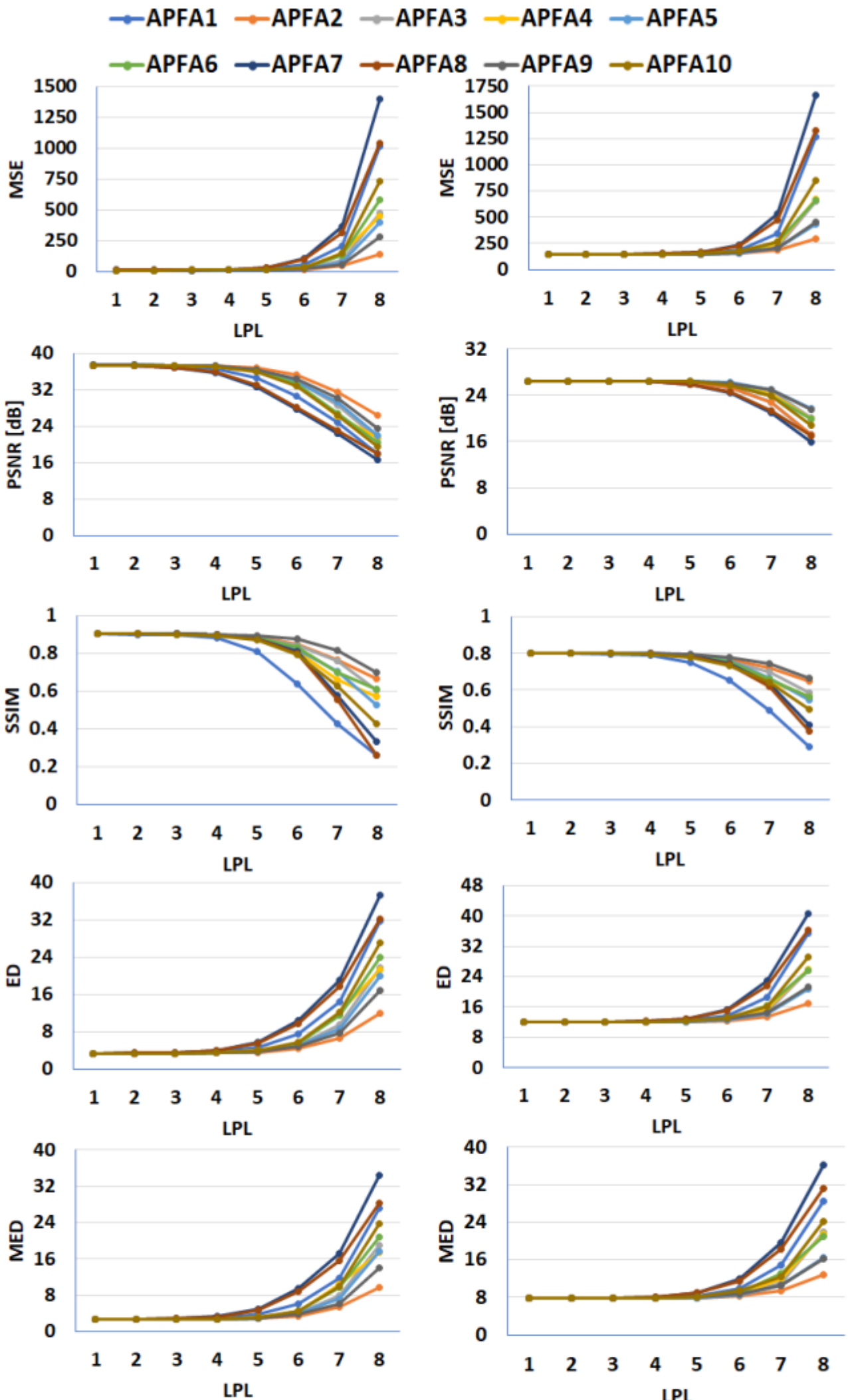
some bits to save power and area.



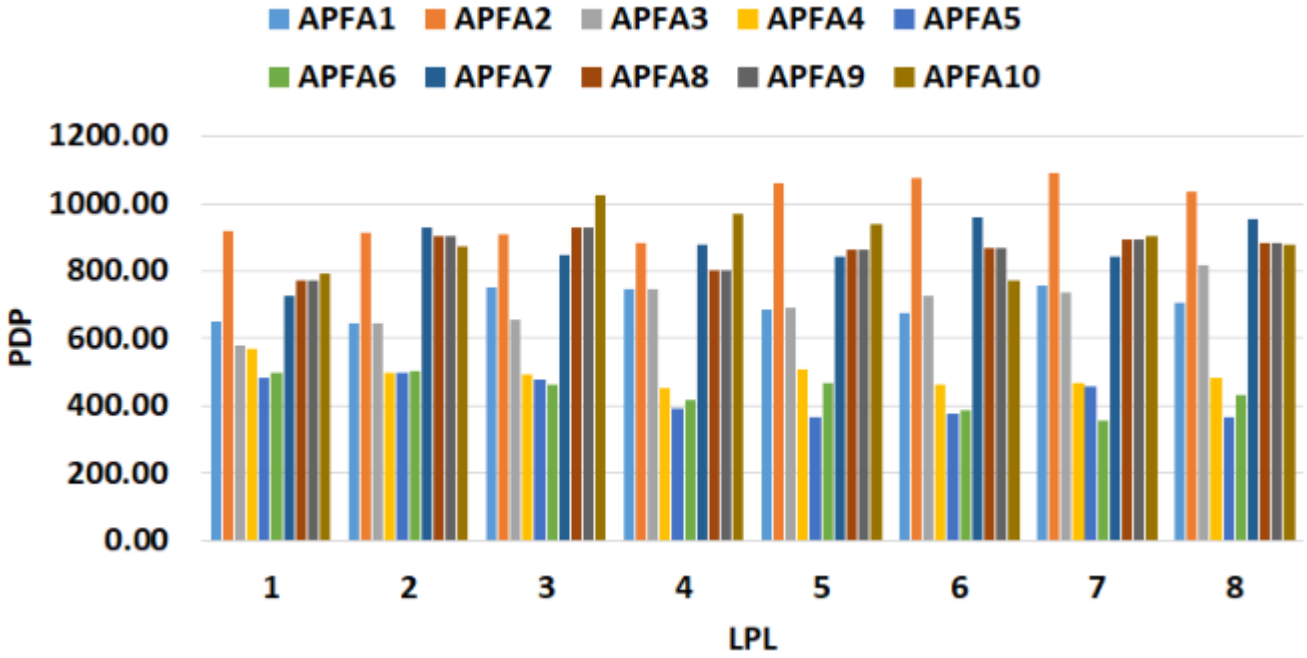
(a)

8. **Figure 7:** Presents two sample raw underwater images used for testing, along with their histograms showing pixel intensity

distributions.


APFA1
APFA2
APFA3
APFA4
APFA5
APFA6
APFA7
APFA8
APFA9
APFA10
MSE
PSNR [dB]
SSIM
ED
MED
LPL

9. **Figure 8:** Graphs evaluation metrics (PSNR, SSIM, etc.) for different approximate adder configurations, demonstrating that up to 5-bit approximation maintains good image quality.



10. **Figure 9:** Compares Power-Delay Product (PDP) values for various approximate adders, identifying the most power-efficient and fastest designs.

11. **Conclusion**
    The study successfully demonstrates that implementing a pipeline Gaussian filter on FPGA with approximate adders significantly improves processing speed (over 150%) and reduces power consumption (over 34%) for underwater image enhancement. Although this comes with increased hardware area and some loss in output precision, the trade-off is acceptable for error-resilient applications like image and video processing. This approach offers a practical solution for real-time underwater imaging systems, enabling better visual quality with efficient hardware use, which is crucial for marine exploration, monitoring, and resource management. The research advances underwater image enhancement by balancing quality, speed, and power in hardware implementations.