Chapter 5

Retinex Based Low-Light Image Enhancement Algorithm and its VLSI Architecture

5.1 Introduction

Real-time low-light image enhancement has several potential applications, such as ADAS, remote sensing, object tracking, etc. Retinex-based algorithms are mostly used to restore the visibility of low-light images. However, they perform complex mathematical operations over a large spatial window. Consequently, their hardware realization is challenging. In this Chapter, a Retinex-based low-light image enhancement algorithm is proposed that employs a low-cost edge-preserving filter for illumination estimation. Although certain approximations are used to curtail the hardware logic resource requirement, the quality of the enhanced image is not compromised.

The organization of the Chapter is as follows. Section 5.2 describes the Retinex model. The proposed LLIE method is described in Section 5.3, and details of its VLSI architecture are presented in Section 5.4. The experimental results and performance analysis are discussed in Section 5.5, and the concluding remarks are presented in Section 5.6.

Contributions of the proposed work

Retinex-based low-light scene enhancement methods efficiently restore images with balanced color and preserved edges. However, Retinex-based low-light image enhancement algorithms involve iterative and complex mathematical operations over a large spatial area. Therefore, the hardware resource requirement of the existing Retinex-based low-light scene enhancement algorithm's FPGA implementations is very high. To address this issue, a Retinex-based low-light image enhancement algorithm and its hardware architecture are presented in this Chapter, which uses the concept of approximate computing to reduce hardware costs. The key contributions of the proposed work are:

- A low-complexity edge-preserving filter is proposed for illumination estimation, which aids in retaining fine details in the decomposed reflectance.
- An approximate computing method is used to implement the exponentiation operation, thereby reducing hardware cost.
- The proposed architecture requires only 10868 LUTs and 7409 registers when implemented on the AMD-Xilinx ZynQ 7 FPGA. Moreover, it can process HD images (1920×1080) at the rate of 60 frames per second (fps).

5.2 The Retinex model

According to the Retinex model [80], an image P is given as the product of illumination E and reflectance R as

$$P(x,y) = E(x,y) \times R(x,y) \tag{5.1}$$

where x and y are the pixel coordinates. The illumination channel E represents the light intensity. The Retinex model assumes that E should be devoid of textures and vary smoothly. On the contrary, the reflectance channel R comprises of textures and details. Estimation of E using the Retinex model should retain edge information and smooth out textural details in it.

5.3 The Proposed Enhancement Algorithm

For an input image P, we obtain the coarse illumination P^{ci} by finding the maximum intensity in a local patch Δ centred at coordinate (x,y) as shown below

$$P^{ci}(x,y) = \underset{(x,y)\in\Delta}{max}\{\underset{c\in(R,G,B)}{max} P^{c}(x,y)\} \tag{5.2}$$

where c is the R, G, B color channel of P. However, the coarse illumination estimated using (5.2) cannot be used for reflectance estimation as it contains blocky effects.

An edge-preserving filter is proposed in [87] for efficient illumination estimation. It preserves sharp edges in the illumination channel and maintains spatial smoothness. This filter performs edge preservation based on guidance image G_i, which is also the maximum channel of P, and it is given as

$$G_i(x,y) = \underset{c\in(R,G,B)}{max}(P^{c}(x,y)). \tag{5.3}$$

G_i retains the edges and imposes a theoretical lower bound on illumination. The filtered output F, which is also the refined illumination, is given as

$$F(x,y) = \frac{\sum_{(m,n)\in\Omega}\beta(m,n)\gamma(m,n)P^{ci}(m,n)}{\sum_{(m,n)\in\Omega}\beta(m,n)\gamma(m,n)} \tag{5.4}$$

where Ω is a local window centered around a pixel in P^{ci}. Here, β and γ are defined as

$$\beta(m,n) = \begin{cases} 1 & P^{ci}(m,n) - G_i(x,y) \geq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.5}$$

$$\gamma(m,n) = exp\left(-\frac{|P^{ci}(m,n) - G_i(x,y)|}{2\sigma^2}\right) \tag{5.6}$$

where σ controls the range similarity. The filter obtains the refined illumination corresponding to a pixel in the guidance image G_i by computing the weighted average of pixels in the coarse illumination P^{ci} over a region Ω.

The region Ω is an odd-size square patch or window. The averaging operation over the entire window would require many multipliers and adders. This increases the total hardware cost. However, to reduce the hardware complexity, we replaced each column of the 2-D window Ω with the minimum value of the coarse illumination pixels occupying the respective column of Ω. This transforms the 2-D filter into a 1-D filter of size 1×r, where r is the number of columns in the given window. The output of the proposed edge-preserving filter is used as the refined illumination $F_m(x, y)$, which is given as

$$F_m(x,y) = \frac{\sum_1^r \beta_m(k)\gamma_m(k)P_{min}^{ci}(k)}{\sum_1^r \beta_m(k)\gamma_m(k)} \tag{5.7}$$

where $P_{min}^{ci}(k)$ represents the minimum value of the k^{th} column of the 2-D window centered at location (x,y) of coarse illumination $P^{ci}(x, y)$. Further, β_m and γ_m are given as

$$\beta_m(k) = \begin{cases} 1 & P_{min}^{ci}(k) - G_i(x,y) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

$$\gamma_m(k) = exp\left(-\frac{|P_{min}^{ci}(k) - G_i(x,y)|}{2\sigma^2}\right). \tag{5.9}$$

Once $F_m(x, y)$ is estimated, reflectance *R(x,y)* can be obtained as given below

$$R^c(x,y) = \frac{P^c(x,y)}{F_m(x,y)} \tag{5.10}$$

where $c \in$ (R,G,B). The main advantages of this method are

- The proposed filter preserves edges and eliminates blocky effects in $F_m(x, y)$ by assigning lower weights to pixels that differ more from the guided image pixels, as can be observed from (5.9).
- The proposed filter limits the lower bound of $F_m(x, y)$ to the maximum channel using (5.8). This further limits the reflectance in the range [0, 1] and prevents an overflow of reflectance values, thereby preserving fine details.

(a) (b) 3×3 (c) 11×11 (d) 31×31

Figure 5.1: Illumination obtained with different sizes of local window. (a) Low-light image. (b)-(d) Illumination with different window sizes.

$F_m(x, y)$ is further modified using the contrast stretch function given by (5.11) to obtain modified illumination $E_{mod}(x, y)$. The contrast stretch function employs the normalized pixel value of $F_m(x, y)$. Finally, $E_{mod}(x, y)$ and $R(x, y)$ are remapped using (5.12) to obtain the visually enhanced image P_E. These operations are mathematically represented as

$$E_{mod}(x, y) = \sqrt{F_m(x, y)}\sqrt{2 - F_m(x, y)}, \tag{5.11}$$

$$P_E(x, y) = E_{mod}(x, y) \times R(x, y). \tag{5.12}$$

The quality of the output image and the hardware resource requirement depend on the size of windows Δ and Ω. Both windows are identical in shape and size. But their region of operation is different. Please note that Δ and Ω operate on G_i and P^{ci}, respectively. To determine an appropriate window size, we experimented with various window sizes and tested the proposed method on several low-light images. Fig. 5.1 shows illumination obtained using different local window sizes. Illumination obtained with a smaller window (3×3) size contains a lot of texture, whereas edge information is lost with a larger window (31×31) size. Fig. 5.1 depicts this fact. The illumination obtained with the 11×11 window size is devoid of textures, and edge information is retained properly. Moreover, the hardware resource requirement for an 11×11 window size is also moderate. Therefore, 11×11 window size was chosen for illumination estimation. Another important aspect of

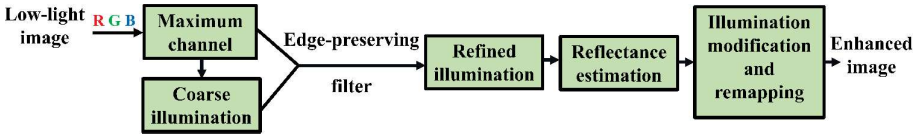


Figure 5.2: Block diagram of the proposed architecture.

the proposed method is boundary handling, which is dealt with by padding the image with stripes of width (r-1)/2, where r is the size of the local window. Further, these stripes are filled with mirror images of boundary elements in the image.

5.4 The Proposed VLSI Architecture

The block diagram of the proposed enhancement architecture is shown in Fig 5.2. First, the maximum channel G_i of the input image P is obtained by finding the maximum intensity in the three color channels. The input pixels, along with G_i, are stored in line buffers. Once the 11^{th} row of G_i is available in the line buffers, a 11×11 size window Δ is centered around each pixel in G_i. The maximum value of each column in Δ is obtained as shown in Fig. 5.3(a). Next, the obtained column maximums are buffered in a register set. Finally, the maximum value of the register set gives the window maximum, i.e., coarse illumination pixel P^{ci}, as shown in Fig. 5.3(b). As the local window slides one pixel to the right, it can be observed from Fig. 5.3(c) that the two consecutive windows differ by only one column. Thus, to find the maximum value within the new window, one needs to find the maximum value within the new column and write it in the register set, which holds the maximum value for each column. The newly added column's maximum value is written in place of the excluded column's maximum value, and the maximum value in the register set gives the maximum value for the newly formed local window. The local window Δ slides in raster scan format over the entire maximum channel G_i, and each local window centered at (x,y) gives the coarse illumination pixel P^{ci} at (x,y), which are also stored in line buffers.

Once sufficient lines of P^{ci} are available in line buffers, a sliding window Ω is centered around each pixel of P^{ci}. The minimum value of the k^{th} column in Ω, i.e., $P^{ci}_{min}(k)$ is

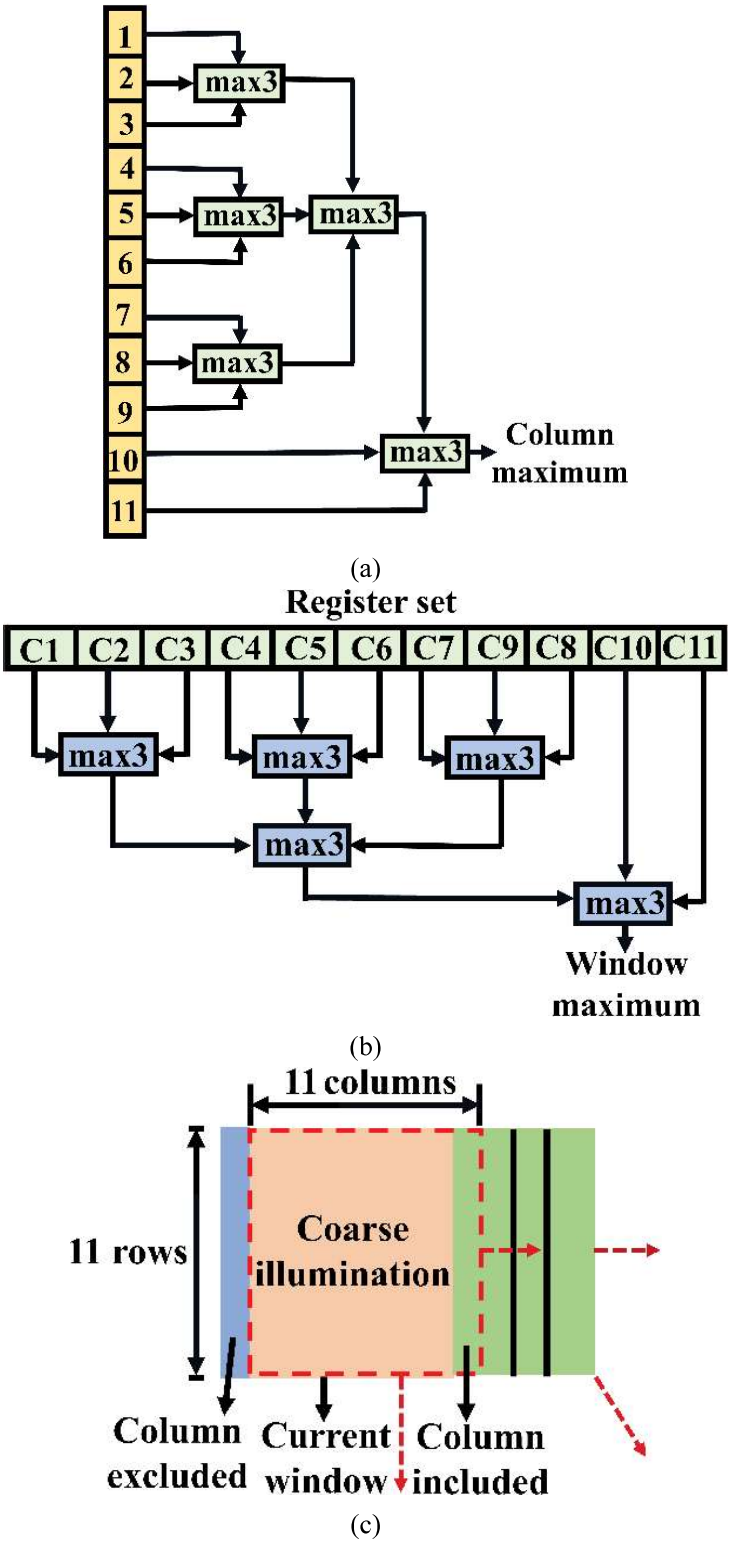
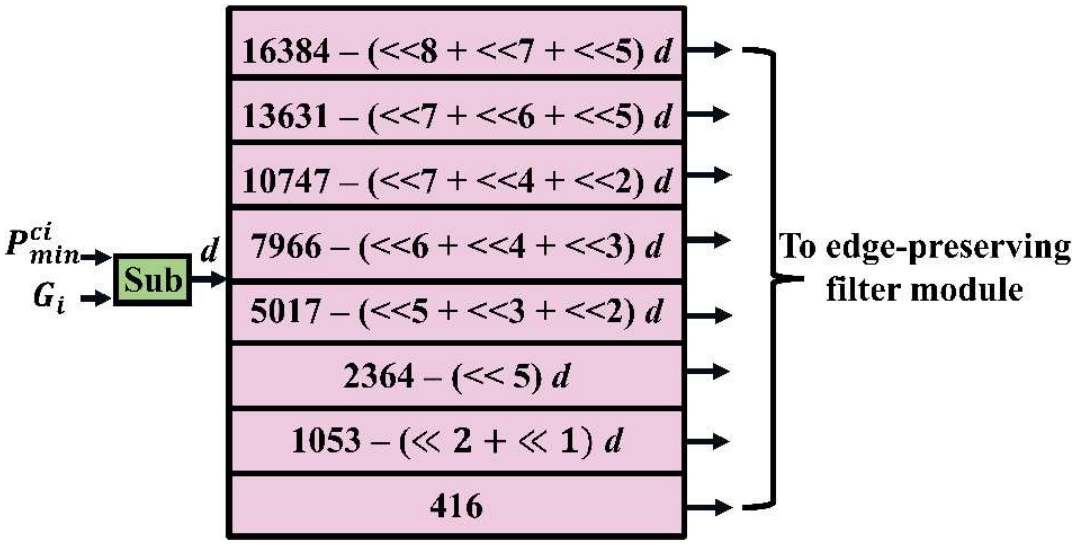


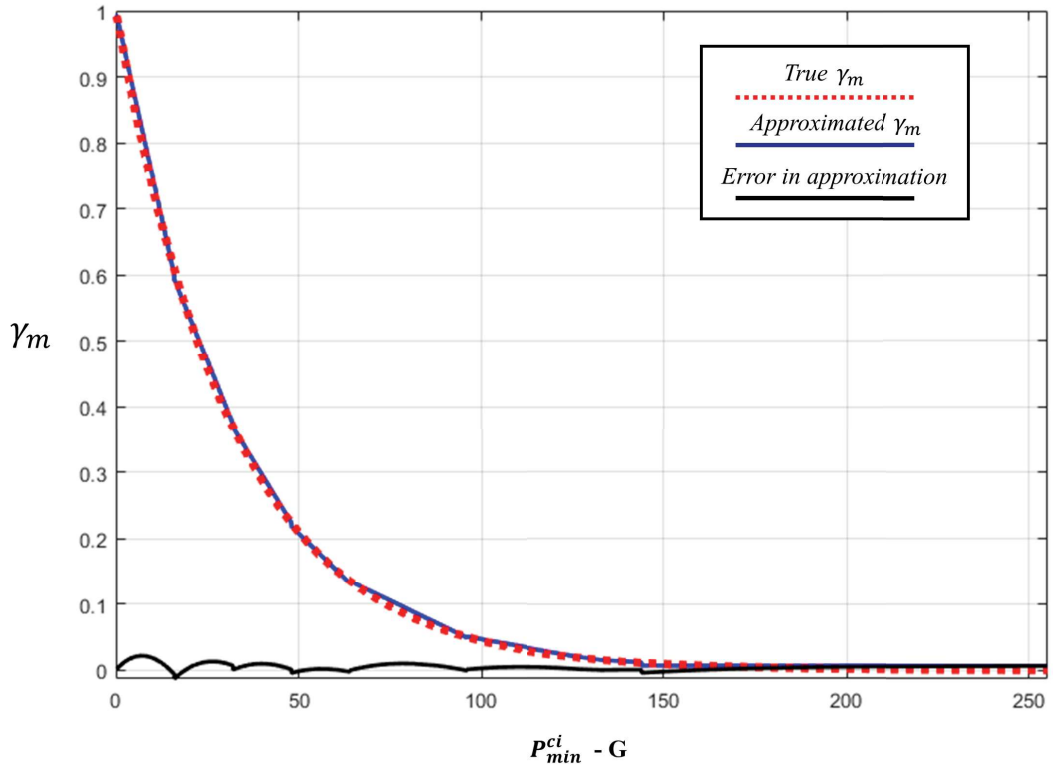
Figure 5.3: Architecture of the local maximum module. (a) Searching the maximum value in a column. (b) Searching the maximum value in a window. (c) Sliding window structure.

computed using the architecture shown in Fig. 5.3(a), replacing the max operation with the min operation. Each $P^{ci}_{min}(k)$ corresponding to a specific position of Ω is stored in a new register set. Next, the difference d between $P^{ci}_{min}(k)$ and G_i is calculated and fed to the linear approximation module to obtain $\gamma_m(k)$ using (5.9). The σ value was set to 4 to attain division operation by simple shifting. To precisely calculate $\gamma_m(k)$ with reduced computational complexity, the $\gamma_m(k)$ function is divided into eight linear segments based on the range of d. A straight-line equation in slope-intercept form represents each segment. Each input to the exponential function is assigned a distinct output value by multiplying the input by the slope of the linear segment and adding the intercept value to it. The slopes of each of the eight linear segments are set so that their multiplication can be achieved by, at most, three distinct left shifts and two addition operations. The linear approximation module outputs eight values of $\gamma_m(k)$ in all eight linear regions for each d. However, depending on the value of d, only one of the eight $\gamma_m(k)$s is useful. The architecture of the linear approximation module is shown in Fig. 5.4(a), which also depicts the slope and intercept values of each linear segment. The slope and intercept have been scaled by 2^{14} to reduce approximation loss. The range of each linear segment is presented in Fig. 5.4(b).

The entire linear approximation module is implemented using shifters and adders. In this way, we have eliminated the requirement for multipliers. A similar exponential approximation technique is used in [108]. However, the exponential function in [108] is divided into several intervals, and pre-computed output values for each interval are stored in LUT-based memory. Depending on the interval in which the input falls, a particular value is sent to the output. It can be observed from Fig. 5.4(b) that the maximum error in $\gamma_m(k)$ is not more than 0.03, even if we adopt the proposed approximation technique. The multiplexer at the input of the edge-preserving filter module, shown in Fig. 5.5, selects an appropriate $\gamma_m(k)$ value depending on the range of d. Next, according to the condition $P^{ci}_{min}(k) \geq G_i$ the weights in the numerator and denominator for each coarse illumination pixel $P^{ci}_{min}(k)$ is calculated using (5.7) and (5.8). It is clear from (5.7) that the sum of the weights in the numerator (N) and denominator (D) is different. So, two N and D adder trees are used in this module, as shown in Fig.5.5. Finally, the output of the N adder tree is divided by the



(a)



(b)

Figure 5.4: (a) Architecture of the linear approximation module. (b) Plot of $\gamma_m(k)$.

output of the D adder tree to obtain the refined illumination pixels $F_m(x, y)$. Here, the division operation is replaced with a multiplier and a lookup table to reduce the hardware resource requirement and improve the latency. Next, based on (5.10), $R(x, y)$ is calculated in all three color channels using $P(x, y)$ and $F_m(x, y)$ obtained from previous modules.

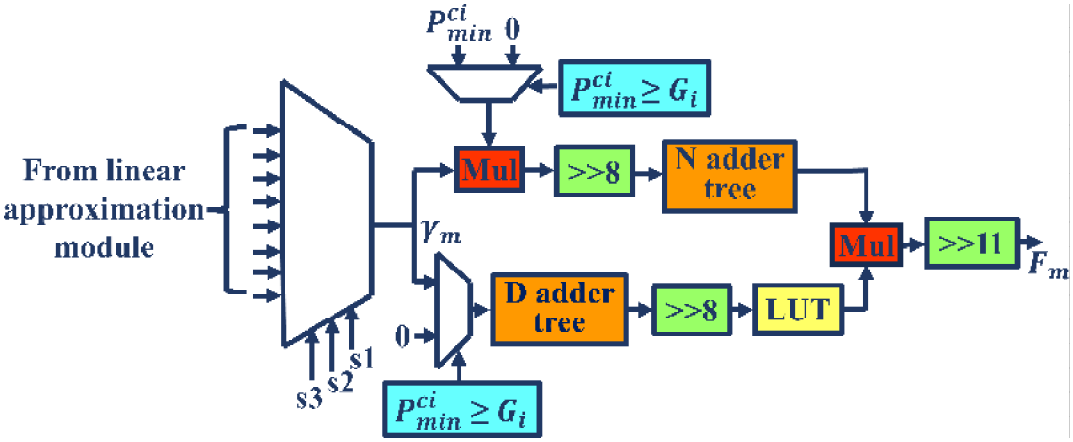
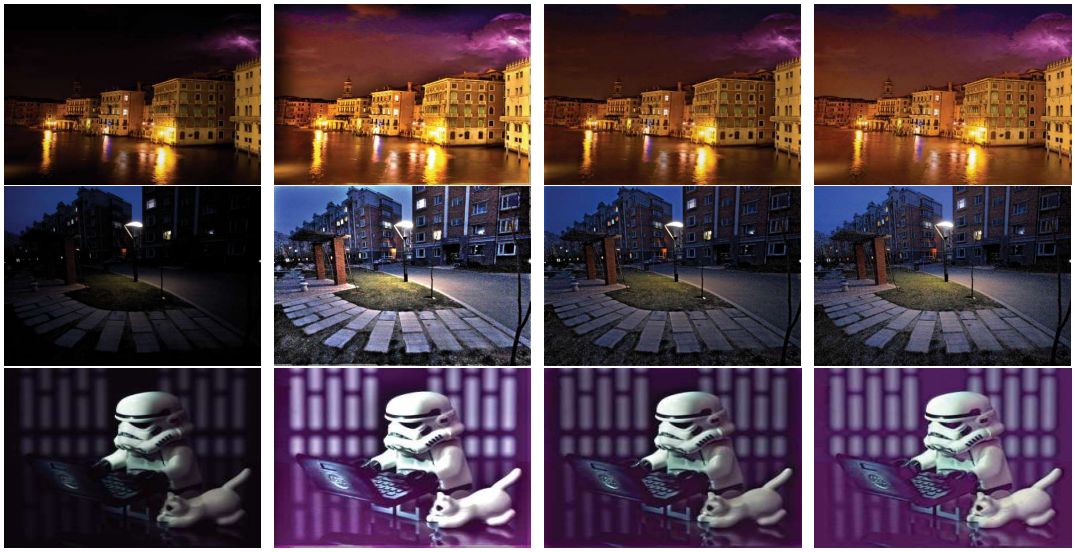


Figure 5.5: Architecture of the edge-preserving filter module.

To simplify the hardware, the complex divider in (5.10) is replaced with a multiplier and a lookup table that stores the probable values of $1/F_m(x, y)$. Next, $F_m(x, y)$ is modified as given in (5.11). To reduce hardware complexity, the nonlinear modification in $F_m(x, y)$ is attained using a LUT-based memory which stores the pre-computed modified values $E_{mod}(x, y)$ stored at the address specified by $F_m(x, y)$. Finally, $E_{mod}(x, y)$ is remapped with $R(x, y)$ to produce an enhanced output image using (5.12). The entire architecture is pipelined to achieve a throughput of one pixel per clock cycle.

5.5 Performance Evaluation and Results Analysis

Experimental results obtained using the proposed method are discussed in this Section. The proposed method's qualitative results are shown in Fig. 5.6. The methods of [85] and [95] were used for visual comparison, as those are also Retinex-based methods. It can be observed from Fig. 5.6 that images obtained using the proposed method are neither over-enhanced nor under-enhanced like others. Moreover, it is clear from Fig. 5.7 that the proposed method effectively limits the range of reflectance in [0, 1] as compared to [95]. Quantitative evaluation of the proposed method was performed using reference metrics peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) on the LOw-Light (LOL) [89] dataset. Higher PSNR and SSIM values signify better restoration and a higher resemblance of the recovered image to the ground truth. DICM [93] and LIME [85]



(a) (b) (c) (d)

Figure 5.6: Visual results (a) Low-light image. (b) Results with [85]. (c) Results with [95]. (d) The proposed method.

(a) (b) (c)

Figure 5.7: Reflectance obtained with different methods. (a) Low-light image. (b) Reflectance with [95]. (c) Reflectance with the proposed method.

datasets were also used for testing the performance of the proposed algorithm using the no-reference metrics natural image quality evaluator NIQE and blind/referenceless image spatial quality evaluator (BRISQUE). A lower value of NIQE and BRISQUE signifies more naturalness in the image. The proposed method was compared with other state-of-the-art methods using these metrics, and the results are presented in Tables 5.1 and 5.2. It is clear from Tables 5.1 and 5.2 that the best PSNR score is obtained with [89], whereas the best SSIM score is obtained with the proposed method. However, the PSNR score of the proposed method is comparable to that of the other methods. Regarding the BRISQUE score, the proposed method outperforms the other methods on the DICM dataset, and it has the second-best score for the LIME dataset. The NIQE score of the proposed method is much better than that of some existing methods.

Table 5.1: Comparison with state-of-the-art methods using PSNR (in dB) and SSIM metrics on the LOL dataset.

	[85]	[89]	[91]	[92]	[93]	The proposed
PSNR↑	17.180	**17.607**	17.551	16.532	15.107	17.021
SSIM↑	0.562	0.654	0.665	0.527	0.564	**0.751**

Table 5.2: Comparison with state-of-the-art methods using NIQE (NIQ) and BRISQUE (BRI) metrics on DICM and LIME datasets.

Methods	**Datasets**			
	DICM		**LIME**	
	NIQ↓	**BRI↓**	**NIQ↓**	**BRI↓**
[85]	4.57	25.25	3.85	**21.53**
[89]	4.19	30.05	4.82	27.03
[91]	3.56	29.31	3.73	23.77
[92]	4.55	36.63	3.66	29.96
[93]	**3.40**	NA	**3.65**	NA
The proposed	3.48	**24.64**	3.67	22.41

Table 5.3: Comparison of hardware resource utilization.

Architectures	[94]	[95]	[96]	[97]	The proposed
LUTs	20043	24176	93989	28362	10868
Registers	420331	22787	-	7795	7409
SRAM	9252kb	3348kb	87kb	358kb	1152kb
DSP48Es	-	13	28	105	22
Frequency (MHz)	200	148.5	100	75.84	148.5
Resolution	640×480	1920×1080	1240×768	1920×1080	1920×1080
Frame rate (fps)	60	60	126	30	60

The hardware architecture of the proposed method has been designed in the AMD-Xilinx Vivado design suite. We used Verilog for implementation and AMD-Xilinx AXI stream IP for video interfacing. The proposed architecture is synthesized on ZynQ7 XC7Z020CLG484-1 FPGA, and its resource utilization is shown in Table 5.3. It is evident from this table that the proposed architecture requires only 45% of LUTs as compared to [94]. Further, the register count of the proposed design is also lower than that of the rest of the designs. Design [94] operates at a high frequency but supports the lowest frame resolution. Additionally, its hardware resource requirement is very high because large-size box filters are used to perform Gaussian blurring. Design [96] achieves the highest frame rate but at a lower resolution. Moreover, a halo-reducing filter is essential in this method, which increases its hardware cost. The same image resolution is supported in designs [95], [97],

and ours. Architecture [95] requires the least number of DSP resources. However, its LUT and register consumption are higher due to the large Gaussian filter used in illumination estimation. Since the operating speed of [97] is lower, it fails to support a high frame rate. The operating frequency of the proposed architecture was set to 148.5 MHz, at which AMD-Xilinx video processing modules operate. This operating frequency is sufficient to process full HD (1920×1080) images at 60 fps.

5.6 Concluding Remarks

In this Chapter, a real-time implementation of a Retinex-based low-light image enhancement algorithm on a ZynQ FPGA is presented. The proposed algorithm produces high-quality output images by employing a low-complexity edge-preserving filter for illumination refinement. The proposed filter efficiently preserves edges and eliminates blocky effects around the edges of the objects in the illumination component. Moreover, it is capable of retaining fine details in the decomposed reflectance. The hardware requirement of the proposed architecture is significantly less than the existing methods, and it can process 60 images of full HD resolution in a second. This proves the candidacy of the proposed method for real-time applications such as ADAS, remote sensing, object detection, etc.