Real-Time Biologically-Inspired Image Exposure Correction

Vassilios Vonikakis, Chryssanthi Iakovidou, and Ioannis Andreadis

Democritus University of Thrace
Department of Electrical & Computer Engineering
Laboratory of Electronics
Xanthi, Greece

Abstract. This chapter presents a real-time FPGA implementation of a biologically-inspired image enhancement algorithm. The algorithm compensates for the under/over-exposed image regions, emerging when High Dynamic Range (HDR) scenes are captured by contemporary imaging devices. The transformations of the original algorithm, which are necessary in order to meet the requirements of an FPGA-based hardware system, are presented in detail. The proposed implementation, which is synthesized in Altera's Stratix II GX: EP2SGX130GF1508C5 FPGA device, features pipeline architecture, allowing the real-time rendering of color video sequences (25fps) with frame sizes up to 2.5Mpixels.

Keywords: FPGA, Real-Time Image Enhancement, Human Visual System, High Dynamic Range Imaging.

1 Introduction

Conventional Standard Dynamic Range (SDR) sensors, which are the case in most consumer-electronics cameras, fail to adequately reproduce HDR scenes, which can be common in outdoor capturing conditions. The main reason for this problem is the low dynamic range of the capturing device, compared to the dynamic range of the scene. As a result, the captured images usually suffer from under/over-exposed regions, in which, little or no information is available to the observer. Adjusting the exposure time is not a solution to the problem, since acceptable reproduction can only be achieved for the dark or the bright image regions but not for both. This is clearly depicted in Fig. 1.

A straight-forward solution to this problem is the use of HDR capturing devices instead of the conventional SDR ones. Nevertheless, HDR cameras cannot always provide a practical solution. Their increased cost has limited their use, while the majority of the existing vision systems are already designed for SDR cameras. Furthermore, an additional algorithm is required in order to perform the mapping of the HDR data to an SDR monitor. Another possible solution is to acquire an HDR image by combining multiple SDR images, captured with different exposures [1]. Nevertheless, this solution can be applied only to static scenes, since it is impossible to have multiple exposures of a moving object, always, at the same background.

Consequently, this approach cannot be applied to time-critical applications i.e. video. A third solution to the above problem is the use of an unsupervised tone-enhancement algorithm, which will compensate for the under/overexposed regions of SDR images, without affecting the correctly exposed ones. This approach overcomes the limitations of the previous two solutions: it does not increase considerably the total cost of the system and it can be used in video sequences as well.

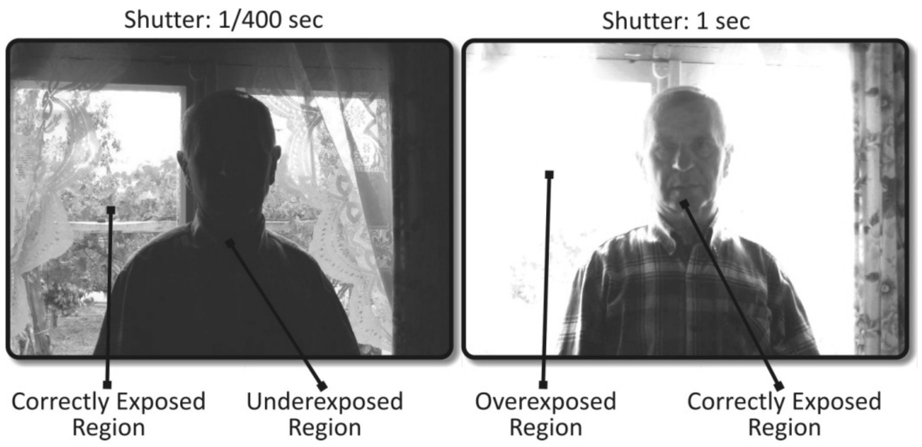

Fig. 1. There is not a single exposure that can adequately capture both the dark and light regions of an HDR scene

Several enhancement algorithms have been proposed in this direction, yet, very few have been implemented in hardware. Some of the most important are the Retinex family of algorithms (Retinex: Retina + Cortex), among which, the more widespread are the "Multi Scale Retinex with Color Restoration" (MSRCR) [2] and the "Variational Framework for Retinex" [3]. The first has been implemented on a Digital Signal Processor (DSP) [4, 5], allowing the real-time, single-scale rendering of grayscale images, with sizes up to 256×256 pixels. A variation of the second has been implemented on an Application Specific Instruction-set Processor (ASIP) [6], allowing the processing of SXGA (1280×768 pixels) or WXGA (1366×768 pixels) still images in 1 sec, or the a real-time rendering of video frames with size 256×256 pixels and frame rates up to 29 frames per second (fps). Both implementations do not meet the VGA standard (color images of 640×480 pixels size and 25fps) in video rendering. Recently, an alternative enhancement algorithm, inspired by the shunting characteristics of the center-surround cells of the Human Visual System (HVS), has been presented in [7]. It exhibits low complexity, as well as the fastest execution times, compared to the algorithms of the previous two implementations. Consequently, it constitutes a good basis for a hardware implementation.

This chapter presents an FPGA implementation of this algorithm. It focuses on the transformations which are necessary in order to optimize the original algorithm for meeting the requirements of an FPGA-based system. The main objective is to implement a pipeline architecture, which will allow the real-time rendering of color video sequences, with sizes greater than the contemporary implementations (256×256 pixels). Two alternative architectures are presented, both synthesized in Altera's Stratix II GX: EP2SGX130GF1508C5 FPGA device. The first allows the real-time (25fps) rendering of color images, with sizes up to 640×480 pixels. The second architecture can render, in real-time, images with sizes up to 2.5MPixels. Both architectures are designed in a way that easily allows future improvements in the core of the algorithm. This, in addition to the FPGA platform, results into a low-cost, robust solution, which can be used to other vision systems as preprocessing, compensating for the low dynamic range of the SDR image sensor.

The chapter is organized as follows. In Section 2 the structure of the original algorithm is briefly described. In Section 3, the transformations of the original algorithm, which will make it suitable for hardware implementation, are presented. The proposed implementations are presented in Section 4, with gate level integrated circuits. The comparison between the hardware and the software are provided in Section 5. Finally, conclusions and a discussion are provided in Section 6.

2 Structure of the Algorithm

In this section the original algorithm, which compensates for the under/over-exposed regions, will be briefly described in order to underline the modifications made in the proposed hardware implementation. A detailed description of the algorithm is out of the scope of this chapter. Extensive details can be found in [7].

The block diagram of the original algorithm is depicted in Fig. 2. In order not to distort the colors of the image, the YCbCr color space is employed, which decorrelates the chromatic and achromatic information. The original method works only on the luminance component and comprises three different stages: a linear stretch of the luminance component, a parameter estimation block and the local enhancement stage. The core of the algorithm is depicted in the equations (1)-(7). Equation (1) stretches linearly the luminance values to the interval $[0,B]$, in order to use the full range of the Y channel. B is the maximum value of the luminance data (255 for 8-bit values), $Y_{\min}$ and $Y_{\max}$ are the minimum and maximum luminance values of the image, while (i, j) denotes the spatial coordinates of the pixel.

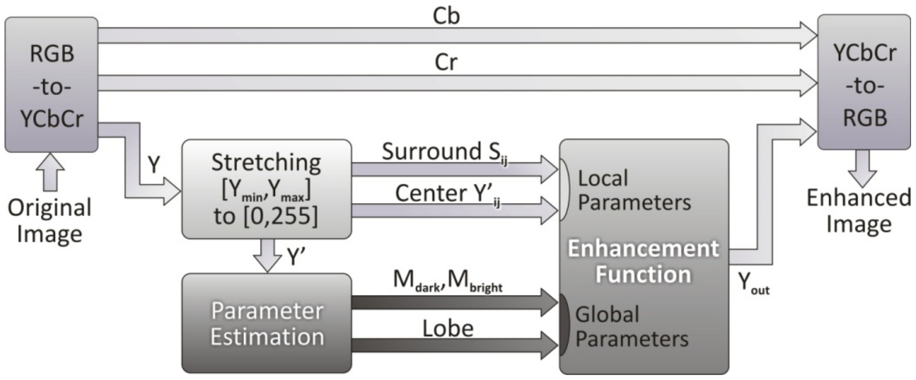

Fig. 2. The block diagram of the original algorithm used in the hardware implementation

$$Y'_{ij} = \frac{Y_{ij} - Y_{\min}}{Y_{\max} - Y_{\min}} \times B \tag{1}$$

$$Y^K_{out,ij}(Y',S) = \begin{cases} \dfrac{\left[B + A\left(S^K_{ij}\right)\right] \cdot Y'_{ij}}{A\left(S^K_{ij}\right) + Y'_{ij}} & \forall S^K_{ij} < \dfrac{B}{2} \\ \dfrac{A\left(S^K_{ij}\right) \cdot Y'_{ij}}{A\left(S^K_{ij}\right) + B - Y'_{ij}} & \forall S^K_{ij} \geq \dfrac{B}{2} \end{cases} \tag{2}$$

$$A(S) = \begin{cases} \left[M_{dark} + q(S)\right] \cdot d(S) & \forall S < \dfrac{B}{2} \\ \left[M_{bright} + q(B-S)\right] \cdot d(B-S) & \forall S \geq \dfrac{B}{2} \end{cases} \tag{3}$$

$$d(x) = \frac{B/2}{B/2 - x} \quad \forall x \in [0, B/2) \tag{4}$$

$$q(x) = \frac{x^2}{Lobe} \tag{5}$$

$$S^K_{ij} = \frac{1}{(2b_K + 1)^2} \sum_{y=i-b_K}^{i+b_K} \sum_{x=j-b_K}^{j+b_K} Y'_{yx} \tag{6}$$

$$Y_{out,ij} = \frac{Y^{K1}_{out,ij} + Y^{K2}_{out,ij} + Y^{K3}_{out,ij}}{3}, \text{ with } b_{K1} < b_{K2} < b_{K3} \tag{7}$$

Equations (2)-(5) describe the enhancement function of the algorithm. As it is clearly depicted in Fig. 2, the enhancement function uses two local and three global parameters. The local parameters, which depend on the local characteristics of the image, are the stretched luminance value Y'_{ij} of the pixel and the average luminance S^K_{ij} of its square surrounding region. K denotes the scale upon which the surround is calculated. Equation (6) describes the surround calculation. Three different surround sizes are employed, with b_K denoting the radius of the square surrounding region for scale K. The final corrected value $Y_{out,ij}$ for every pixel, is calculated by equation (7) and it is the average between the corrected values $Y^{K}_{out,ij}$ of the three spatial scales. The global parameters, which depend on the global image statistics, are M_{dark}, M_{bright} and *Lobe*. Their exact calculation is described by the following equations.

$$bin\_low = \frac{\sum_{i=1}^{py}\sum_{j=1}^{px} u\left(\frac{B}{3} - Y'_{ij}\right)}{px \cdot py} \times 100 \tag{8}$$

$$M_{dark} = \frac{270}{100}\left(100 - bin\_low\right) + 30 \tag{9}$$

$$bin\_high = \frac{\sum_{i=1}^{py}\sum_{j=1}^{px} u\left(Y'_{ij} - 2\frac{B}{3}\right)}{px \cdot py} \times 100 \tag{10}$$

$$M_{bright} = \frac{270}{100}\left(100 - bin\_high\right) + 30 \tag{11}$$

$$bin\_middle = 100 - bin\_low - bin\_high \tag{12}$$

$$Lobe = \frac{29}{100}\left(100 - bin\_middle\right) + 1 \tag{13}$$

where *px*, *py* are the dimensions of the original image and $u(\cdot)$ is the unitary step function. *bin_low*, *bin_middle* and *bin_high* are the bins of a 3-bin normalized histogram (*bin_low*[0, *B*/3], *bin_middle*(*B*/3, 2*B*/3), *bin_high*[2*B*/3, *B*]) that divides the range of the stretched luminance channel Y' into 3 equal tone intervals: dark, medium and bright. For a detailed analysis on the characteristics of the global parameters, refer to [7].

3 Algorithm Optimization

3.1 Optimizing the Structure of the Algorithm

The straight-forward conversion of an algorithm's software implementation to hardware, usually leads to unsatisfactory results. Most of the times many transformations are necessary in order to optimize the algorithm for hardware implementation. This section focuses on these optimizations. The original algorithm, as depicted in Figure 2, requires three different scans of the image, in order to get the final result. This is depicted in Figure 3.

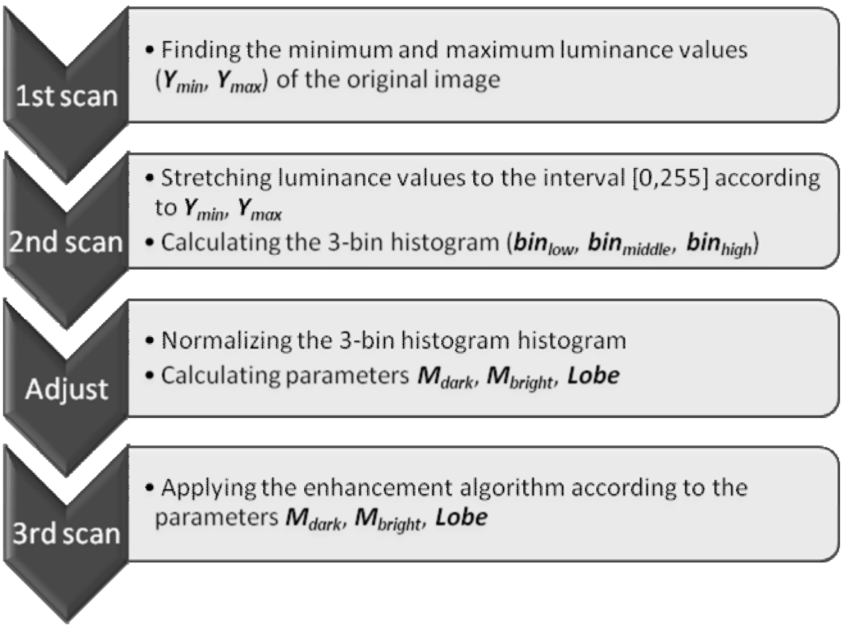
1st scan
• Finding the minimum and maximum luminance values (Y_{min}, Y_{max}) of the original image
2nd scan
• Stretching luminance values to the interval [0,255] according to Y_{min}, Y_{max}
• Calculating the 3-bin histogram (bin_{low}, bin_{middle}, bin_{high})
Adjust
• Normalizing the 3-bin histogram histogram
• Calculating parameters M_{dark}, M_{bright}, $Lobe$
3rd scan
• Applying the enhancement algorithm according to the parameters M_{dark}, M_{bright}, $Lobe$

Fig. 3. The structure of the original software implementation

The first scan of the image is necessary in order to find Y_{min} and Y_{max}. In the second scan, equation (1) is applied to all pixels, stretching their luminance values to [0,255]. The second scan can be eliminated using a Look-Up-Table (LUT). Instead of applying the stretching transformation to all image pixels separately, equation (1) can be executed only 256 times, one for each luminance value. These precomputed values are stored in the "Stretching LUT". Feeding the original luminance value Y_{ij} of a pixel to the *StretchingLUT* module, will output its stretched luminance value Y'_{ij}, as equation (14) indicates.

$$Y'_{ij} = StretchingLUT\left[Y_{ij}\right] \tag{14}$$

This simple transformation reduces the required scans of the image to two, as Figure 4 indicates.

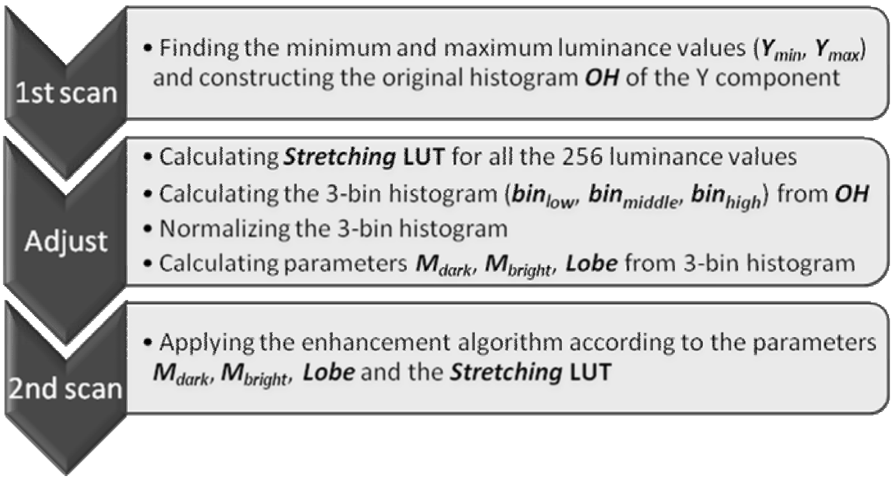
1st scan
• Finding the minimum and maximum luminance values (Y_{min}, Y_{max}) and constructing the original histogram OH of the Y component
Adjust
• Calculating Stretching LUT for all the 256 luminance values
• Calculating the 3-bin histogram (bin_{low}, bin_{middle}, bin_{high}) from OH
• Normalizing the 3-bin histogram
• Calculating parameters M_{dark}, M_{bright}, Lobe from 3-bin histogram
2nd scan
• Applying the enhancement algorithm according to the parameters M_{dark}, M_{bright}, Lobe and the Stretching LUT

Fig. 4. The structure of the algorithm after the Stretching LUT transformation

3.2 Requirements for a Pipelined Architecture

Implementing a pipelined architecture is a primary objective, since it will allow a high throughput, which is essential for real-time applications. For this reason, two identical modules are needed for the original histogram generator (*OH1*, *OH2*), the stretching LUT (*StretchingLUT1*, *StretchingLUT2*) and the global parameters (*Parameters1*, *Parameters2*). The first modules process the odd frames, while the second modules process the even ones. This is depicted in Figure 5. When *frame k* is in the *adjust* state, the *1st scan* is performed for *frame k+1*. Consequently, *OH1* processes *frame k*,

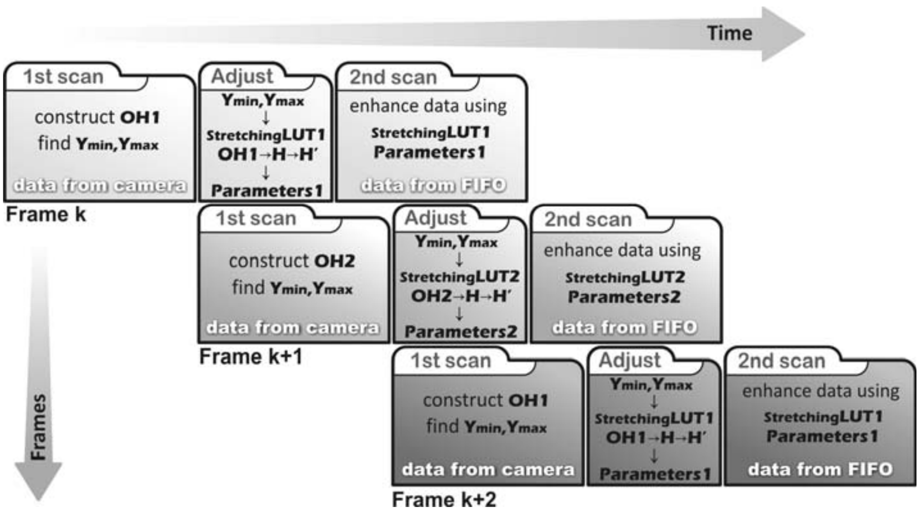

Fig. 5. Double components are necessary in order to achieve a pipelined architecture

while *OH2* generates the histogram of *frame k+1*. When *frame k* is in the *2nd scan*, *frame k+1* is in the *adjust* state. Then, *StretchingLUT1* processes *frame k*, while *StretchingLUT2* is being loaded with the stretching values for *frame k+1*. Similarly, when *frame k* is in the *2nd scan*, its image data are enhanced using the global parameters from module *Parameters1*. At the same time, the global parameters of *frame k+1* are calculated using the module *Parameters2*.

4 Hardware Implementation

This section presents the key stages of the proposed hardware implementation. Figure 6 depicts the block diagram of the system. We assume that the frames are fed into the FPGA sequentially, pixel by pixel, in the RGB color space format. Consequently, in order to create the second scan, the data are fed into a FIFO memory, after the transformation from RGB to YCbCr. This FIFO should have an appropriate length in order to introduce a delay to the data, equal to the execution time of both *1st scan* and *adjust* stages. Taking into consideration that the adjust stage requires 300 clock cycles, the FIFO length should comprise 1frame+300 memory elements. As a result, when the *2nd scan* is about to begin, the YCbCr data will be ready for processing.

As mentioned in the previous section, two *StretchingLUT* modules are required for a pipelined architecture: one for the odd and one for the even frames. Figure 6 however

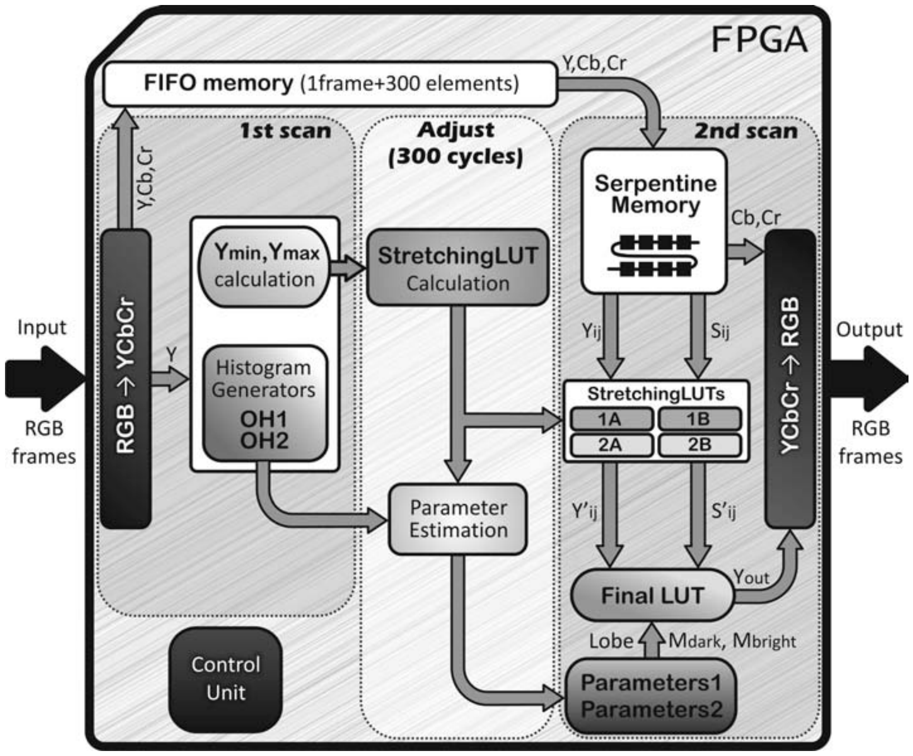

Fig. 6. The proposed hardware implementation

shows that two extra *StretchingLUT* modules are required: one for the luminance value of the pixel Y_{ij} and one for its surround luminance S_{ij}. This increases the total number of modules to four. *StretchingLUT* 1A and 1B are identical and are used simultaneously for the luminance values Y_{ij} and S_{ij} of the odd frames. Similarly, *StretchingLUT* 2A and 2B are also identical and are used simultaneously for the luminance values Y_{ij} and S_{ij} of the even frames. While the odd LUTs are used to transform the luminance values, the even LUTs are loaded with data. In the following frame, the even LUTs are used for transforming luminance values and the odd LUTs are stored with data.

4.1 Color Space Transforms

The transformations RGB→YCbCr and YCbCr→RGB are the first and last processing stages of the system. The original mathematical forms of the transformations comprise floating point arithmetic, which can be computationally intensive in a straight-forward implementation. In order to avoid the floating point operations, the transformations are altered as follows:

$$\begin{aligned}\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} &= \begin{bmatrix} 0.230R + 0.661G + 0.109B \\ -0.101R - 0.338G + 0.439B + 128 \\ 0.439R - 0.399G - 0.040B + 128 \end{bmatrix} = \\ &= \frac{1}{1024}\begin{bmatrix} 236R + 677G + 112B \\ -103R - 346G + 450B + 131072 \\ 450R - 409G - 41B + 131072 \end{bmatrix}\end{aligned} \tag{15}$$

$$\begin{aligned}\begin{bmatrix} R \\ G \\ B \end{bmatrix} &= \begin{bmatrix} 1.084Y + 1.793(Cr - 128) \\ 1.084Y - 0.534(Cr - 128) - 0.213(Cb - 128) \\ 1.084Y + 2.115(Cb - 128) \end{bmatrix} = \\ &= \frac{1}{1024}\begin{bmatrix} 1110Y + 1836Cr - 235012 \\ 1110Y - 547Cr - 218Cb + 97911 \\ 1110Y + 2166Cb - 277217 \end{bmatrix}\end{aligned} \tag{16}$$

The floating point numbers are multiplied by 1024 and the appropriate divider is introduced at the front of the matrix. The number 1024 is selected for two reasons. First, the multiplication with any integer number greater or equal to 1000 results to another integer number, thus, avoiding the decimal points of the original transformation. Second, 1024 is a power of 2 (2^{10}=1024) and, therefore, the divider can be implemented with 10 right shifts of the final result. The implementation of equations (15) and (16) is depicted in Figure 7 and Figure 8, respectively.

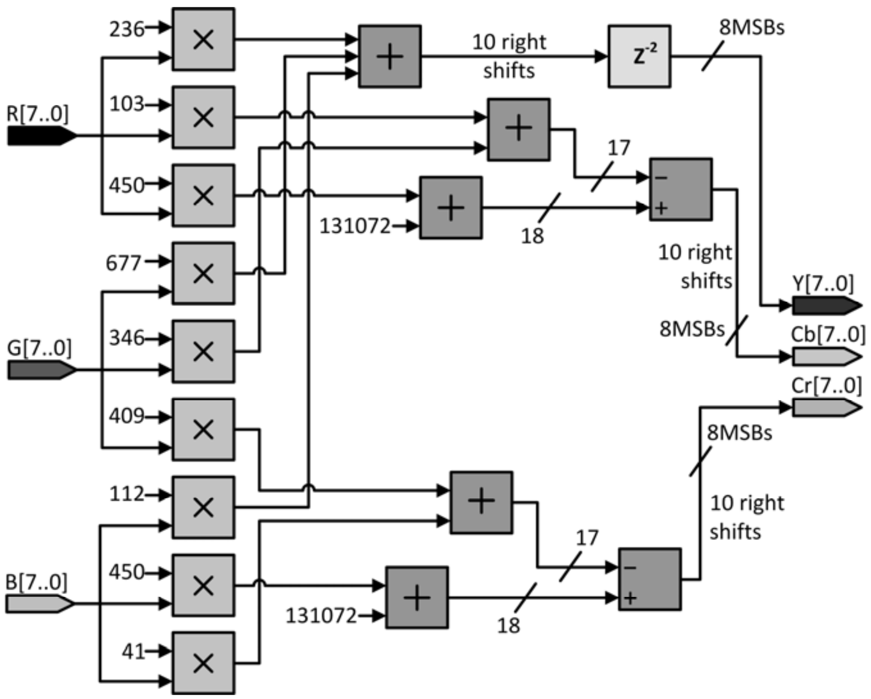

Fig. 7. Implementation of the RGB→YCbCr transformation

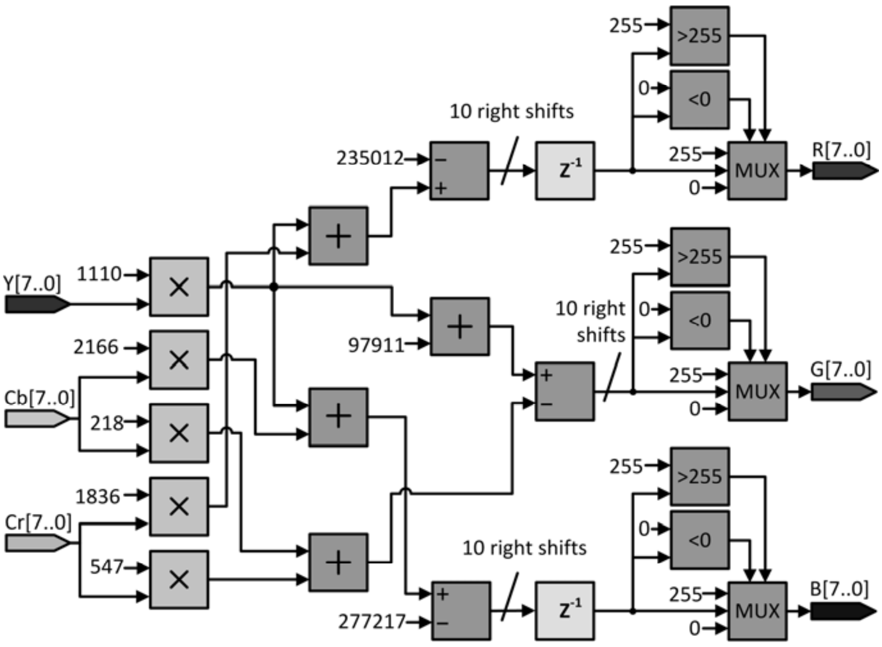

Fig. 8. Implementation of the YCbCr→RGB transformation

The above implementations employ many multiplications, which are considered to be computationally intensive operations. For this reason these multiplications are implemented with parallel shifts, as the following example indicates.

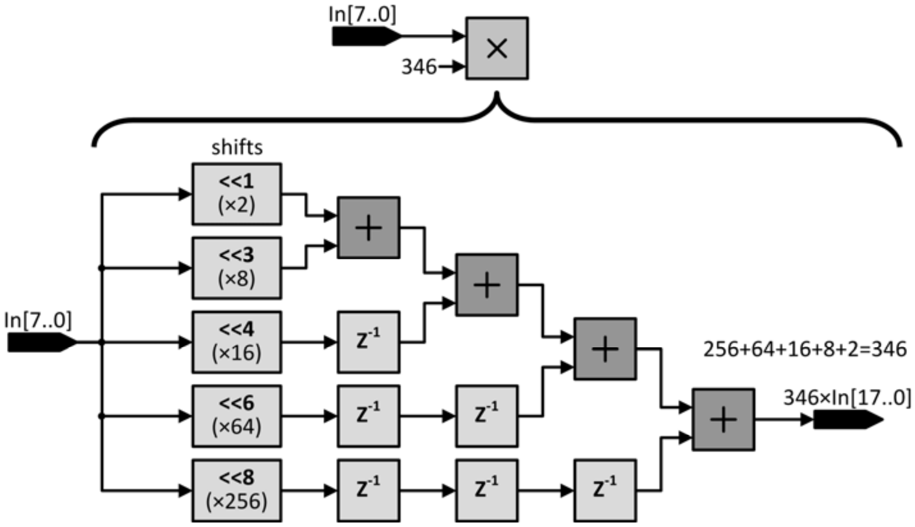

Fig. 9. Implementation of one of the multiplications of Figures 7 and 8

4.2 Calculation of StretchingLUT Modules

The stretching function of equation (1) comprises among others, a multiplication and a division. A straight-forward implementation of these operations would be inefficient, since they are considered expensive in terms of recourses. For this reason, equation (1) is implemented in an incremental way.

$$Stretching\left(x\right) = Stretching\left(x-1\right) + \frac{255}{Y_{\max} - Y_{\min}} \forall x \in \left[Y_{\min}, Y_{\max}\right] \tag{17}$$

The idea behind this alternative approach is that the stretching transformation divides the luminance channel into equal increments whose size is determined by the incremental factor 255/(Y_{max}–Y_{min}). Consequently, every stretched value differs from the previous and from the next by the incremental factor. This means that a LUT could store all the possible incremental factors and feed the appropriate to an accumulator, which would calculate the stretched luminance values. This is depicted in Figure 10a.

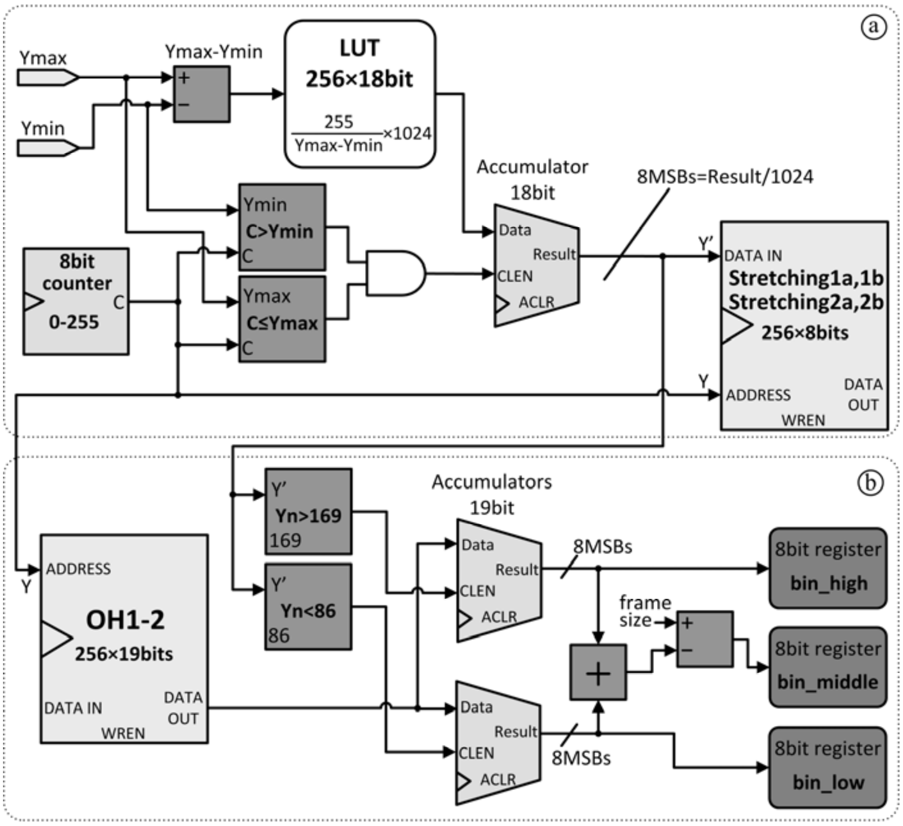

Fig. 10. a. Calculation of the StretchingLUTs. b. Calculation of the 3-bin histogram.

Similarly to the previous subsection, in order to avoid floating point arithmetic and maintain the accuracy of the calculations, the LUT stores the incremental factors multiplied by 1024. The final result is calculated by 10 right shifts (division by 1024). This is depicted in the following equations.

$$accumulator(x) = accumulator(x-1) + \frac{255\times1024}{Y_{\max} - Y_{\min}} \forall x \in [Y_{\min}, Y_{\max}] \quad \text{with } accumulator(Y_{\min}) = 0 \tag{18}$$

$$StretchingLUTs(x) = \frac{accumulator(x)}{1024} \tag{19}$$

4.3 Global Parameter Calculation

The global parameters, which are necessary for the main enhancement function, depend upon a 3-bin histogram of the stretched luminance component, as equations (8)-(13) indicate. Figure 10b depicts their implementation. The stretched luminance value Y' is driven into two comparators, while the original luminance value Y is driven into the address bus of the appropriate original histogram (OH1 for odd frames or OH2 for even frames). The data output of the original histogram is sent to two accumulators. Depending on the output of the two comparators, the corresponding accumulator is activated and sums the number of pixels having the current Y' luminance value. When all the luminance values of the interval [Y_{min}, Y_{max}] have been processed, the three registers, that are depicted in Figure 10b, will store the number of pixels of the stretched luminance component that belong to the high bin (light tones), the middle bin (mid-tones) and the low bin (dark tones). Despite the fact that the accumulators are 19-bit wide, only the 8 most significant bits are registered, since high precision is not vital for global image statistics.

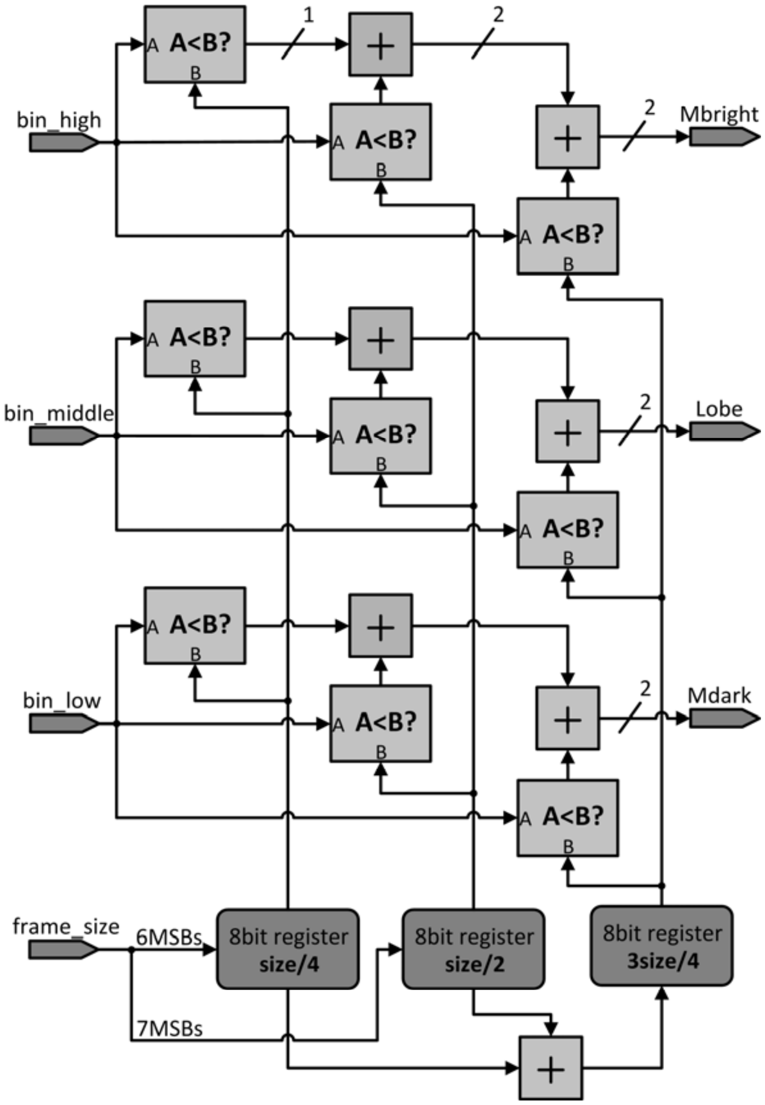

Fig. 11. Implementation of the global parameters

Figure 11 depicts the proposed implementation for the calculation of the global parameters. The 3-bin histogram is correlated with the desired values of the global parameters. These are 2-bit parameters which can have four different values. Their values are inversely proportional to the percentage of image pixels that a particular bin has. If for example *bin_low* occupies more than ¾ of the image size, the value of M_{dark} will be 00. On the contrary, if *bin_low* occupies less than ¼ of the image size, M_{dark} will be 11. The same associations hold for *bin_middle* with the *Lobe* parameter and *bin_high* with M_{bright}.

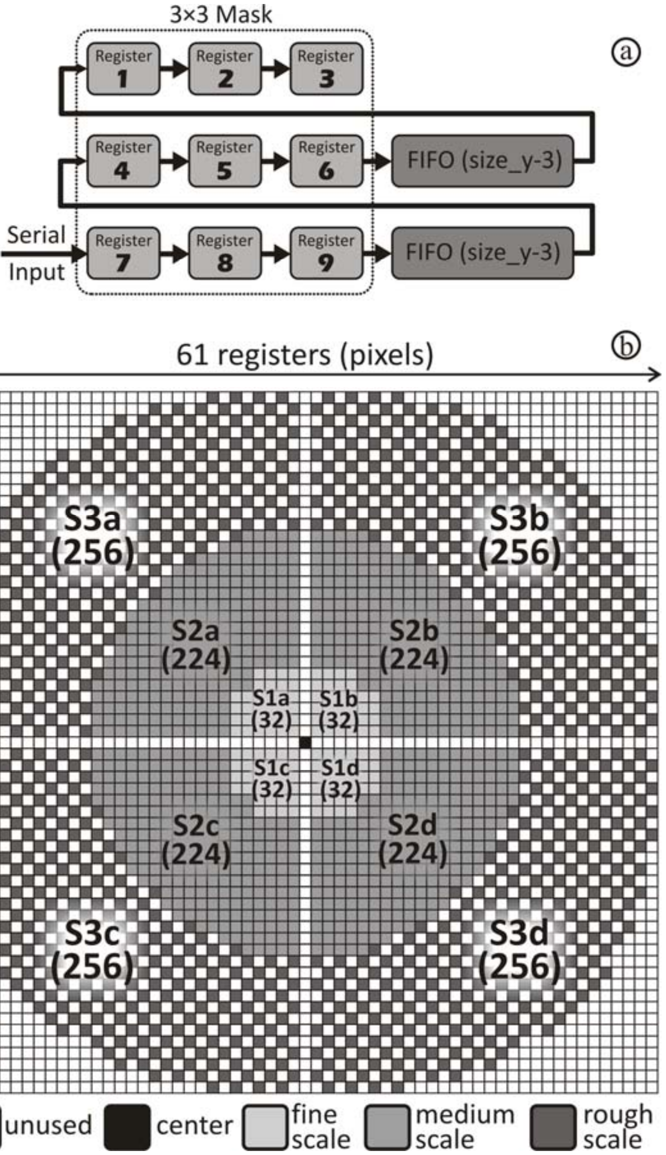

Fig. 12. a. A 3×3 serpentine architecture. b. The proposed serpentine architecture.

4.4 Surround Calculation

The average surrounding luminance of every pixel is a key local parameter of the main enhancement function. Its calculation introduces a considerable challenge: the surround calculation is principally a parallel operation, since many pixel values have to be averaged instantly, while the pixel values are fed to the system sequentially from the camera. For this reason, a serpentine memory architecture [8] is employed, allowing parallel access to many pixel values. Figure 12a depicts a 3×3 serpentine architecture.

The original enhancement algorithm requires large surround sizes of three different spatial scales. In order to implement this, a wider serpentine architecture is employed. The whole layout of the serpentine memory consists of 61×61, 24-bit registers and (60 × *py*)-61 wide FIFOs, (where *py* is the width of the picture). Five different types of registers are located in the proposed mask. At every clock cycle the central register contains the original luminance value of the pixel to be enhanced. The registers that appear white in Figure12b are not used in the computation, while the three different concentric areas of registers S_1, S_2, S_3 represent the three scales of surrounding neighborhoods. The number of registers in each spatial scale has been carefully selected in order to be a power of 2. This bypasses the expensive divisions, which are required for averaging, by using right shifts in the final result.

$$S_1 = S_1a + S_1b + S_1c + S_1d = 128\,pixels \tag{20}$$

$$S_2 = S_1 + S_2a + S_2b + S_2c + S_2d = 1024\,pixels \tag{21}$$

$$S_3 = S_2 + S_3a + S_3b + S_3c + S_3d = 2048\,pixels \tag{22}$$

$$Surround_1 = \frac{S_1}{128} \quad \text{7 right shifts} \tag{23}$$

$$Surround_2 = \frac{S_2}{1024} \quad \text{10 right shifts} \tag{24}$$

$$Surround_3 = \frac{S_3}{2048} \quad \text{11 right shifts} \tag{25}$$

The third scale (S_3) is different than the other two. Some of the registers participating in the computation are symmetrically scattered, in order to minimize the number of needed registers and cover a wider area of pixels. In order to sum at the same time the registers of the serpentine, large parallel adders are employed as Figure 13 depicts.

The final multiscale surround value is obtained by averaging the three surround values as follows:

$$S = \frac{2 \times S_1 + S_2 + S_3}{4} \tag{26}$$

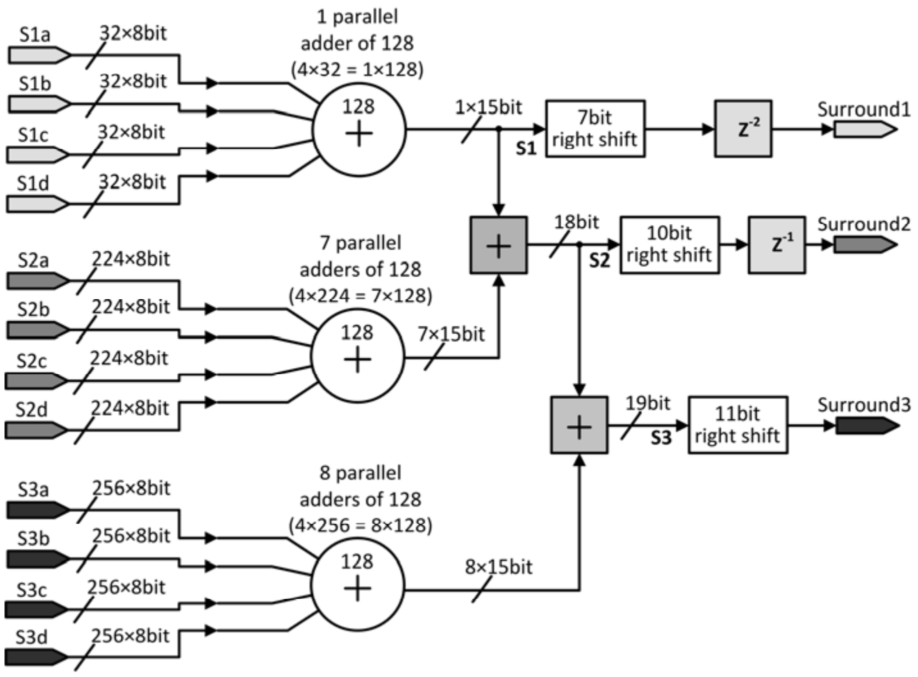

Fig. 13. Parallel summation of the serpentine registers

For every pixel (i,j) its luminance value Y_{ij} and its multiscale surround value S_{ij} are fed to the *StretchingLUT* modules, in order to get the stretched values. This is depicted in Figure 14.

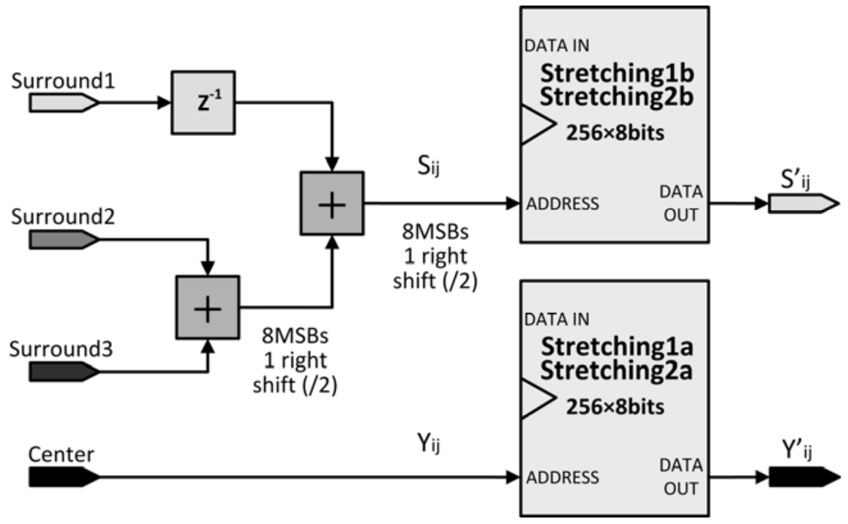

Fig. 14. Calculation of the local parameters

4.5 Enhancement Function

The enhancement function, as described by the equations (2)-(5), comprises several divisions and multiplications, which are expensive in terms of resources. In order to bypass the use of dividers and multiplicators, the enhancement function is stored into a LUT. This *FinalLUT* module, shown in Figure 6, is a ROM which is addressed with the four parameters of the main enhancement function and outputs at every clock cycle the equations' result. Figure 15 depicts the addressing of the *FinalLUT* module.

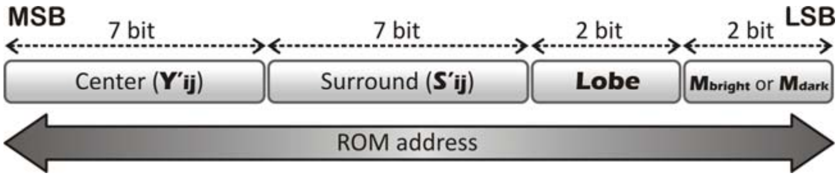

Fig. 15. Addressing of the *FinalLUT*

A reduction of the precision of the local parameters (7bits instead of 8bits) is introduced, in order to maintain the size of the *FinalLUT* within the ranges of current FPGA technology. The size of the ROM memory used is $2^{18}\times8$ bits = 256KB. As it will be shown later, this precision reduction does not affect the image quality of the final output of the system. Apart from resource efficiency, the use of the LUT allows rapid future improvements to the system. This can be done by simply changing the data of the LUT, instead of redesigning the whole system.

4.6 Alternative Implementation

As the frame sizes increase, the FPGA's memory resources become inadequate, and the FIFO memory that was used to create the second scan must be relocated outside the FPGA, as an external memory. An external memory is a costly component to include in a hardware implementation. Furthermore, it increases considerably the complexity of the system.

The only reason for the existence of the FIFO memory is the requirement to find the minimum and maximum luminance values of every frame. In a video sequence however, adjacent frames usually present small differences, since not much can change in 1/25 of a second. In fact, the global statistics of the frames remain practically unchanged. Taking this into consideration, the first stage of the hardware implementation described in the previous sections can be omitted. This can be done by using the global parameters M_{dark}, M_{bright} and Lobe and the *StretchingLUTs* of the previous frame. The pipelining structure of this alternative architecture is depicted in Figure 16.

As it is shown in Figure 16 only one scan for every frame is needed. Frame k enters the FPGA and is enhanced using the *StretchingLUT* and the global parameters (M_{dark}, M_{bright} and *Lobe*) of frame k-2. At the same time, its global statistics (histogram, Y_{max} and Y_{min}) and the parameters of frame k-1 (*StretchingLUT*, M_{dark}, M_{bright} and *Lobe*) are computed, in order to be used for the enhancement of the next frame. This implementation presents the highest performance in terms of execution time and frame size.

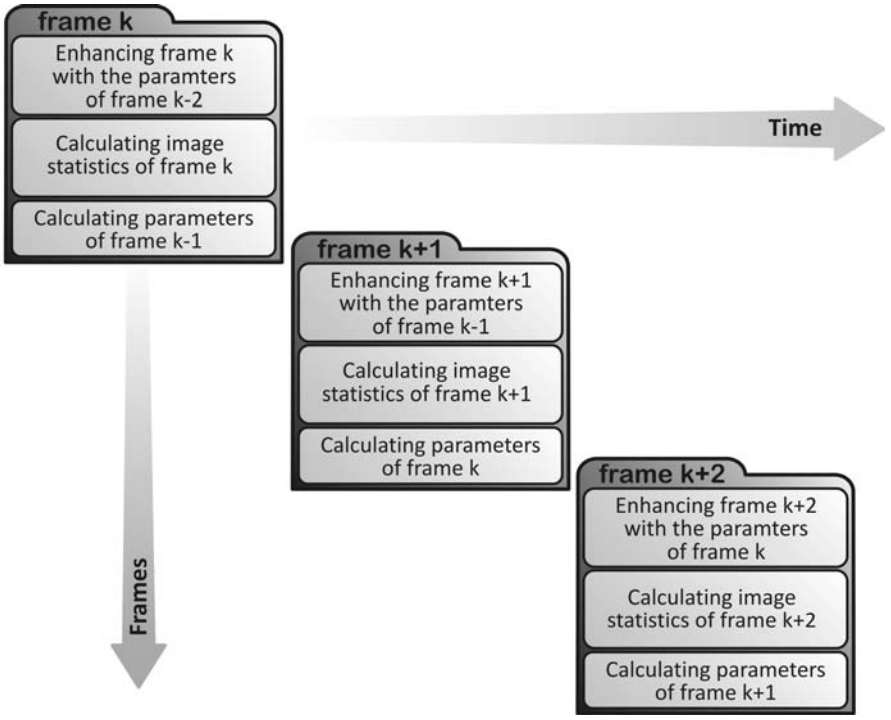
frame k
Enhancing frame k with the paramters of frame k-2
Calculating image statistics of frame k
Calculating parameters of frame k-1
Time
frame k+1
Enhancing frame k+1 with the paramters of frame k-1
Calculating image statistics of frame k+1
Calculating parameters of frame k
frame k+2
Enhancing frame k+2 with the paramters of frame k
Calculating image statistics of frame k+2
Calculating parameters of frame k+1
Frames

Fig. 16. The pipelining procedure of the alternative implementation

5 Hardware and Software Comparison

The accuracy reduction of the local parameters inevitably induces errors, compared to the software implementation of the algorithm. Figure 17 depicts the results of an error

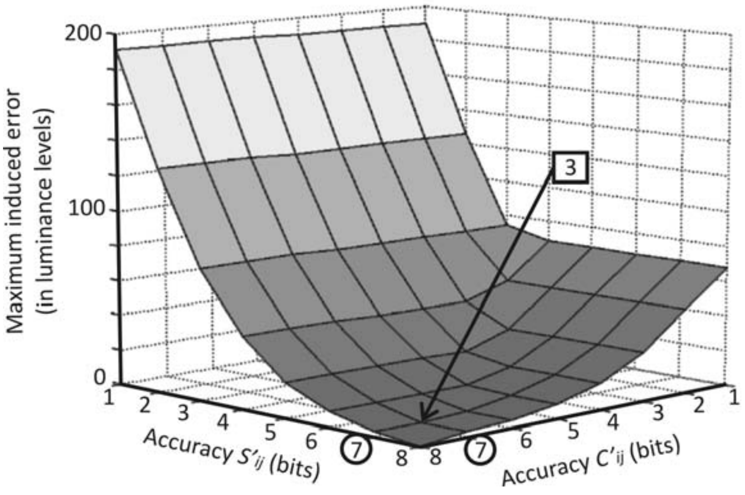

Fig. 17. Error analysis for the accuracy reduction in local parameters

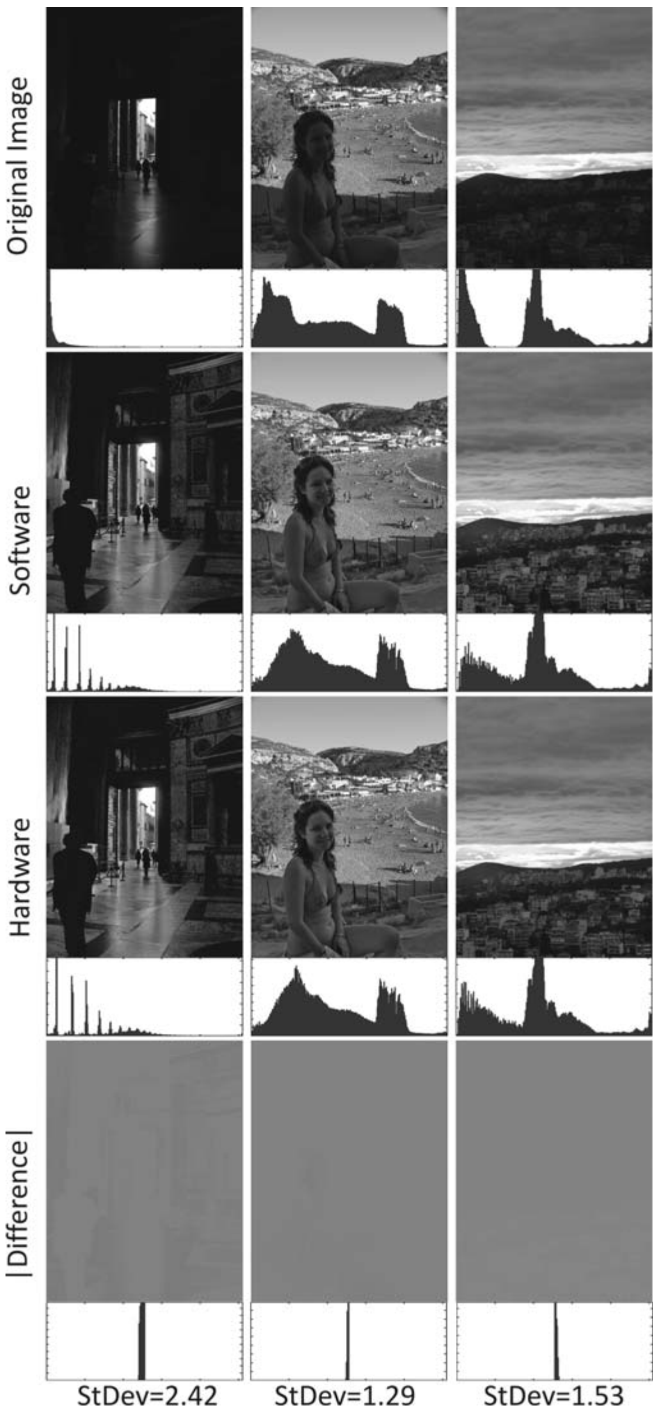

Fig. 18. Comparison between hardware and software

analysis for the accuracy reduction of the local parameters. This analysis shows that the maximum induced error can only be of three gray levels. A human observer is not capable of discriminating such errors, especially in video sequences. This conclusion is also confirmed in Figure 18, which depicts a visual comparison between the two results, as well as the standard deviation of their absolute difference.

6 Discussion and Conclusions

The proposed hardware architectures were synthesized in Altera's Stratix II GX: EP2SGX130GF1508C5 FPGA device. Table 1 depicts the simulation results from Altera Quartus II 5.1 CAD tool.

Table 1. Simulation results in Altera Quartus II 5.1 CAD tool

Altera Stratix II Simulation Results			
	Original Implementation		**Alternative Implementation**
	Color (24bit)	Grayscale (8bit)	Color (24bit)
Frame size	400×400 pixels	640×480 pixels	2.5 MPixels
Frame rate	25 fps	25 fps	25 fps
Total ALUTs	50,037/106,032 (47%)	44,034/106,032 (41%)	49,763/106,032 (47%)
Total registers	43,873	43,873	43,793
Total memory bits	5,492,736/ 6,747,840 (81%)	5,099,522/ 6,747,840 (75%)	2,609,151/ 6,747,840 (39%)
Total Combinational Functions	19,405	18,476	18,841
DSP Blocks	0	0	0
Maximum Frequency	66.66 MHz		

For both implementations, the maximum frequency of the system is 66.66MHz. This frequency is determined by the slowest module of the system, which are the parallel adders in the surround calculation. The original implementation allows the real-time rendering of color frames with size 400×400 pixels, or grayscale frames with size 640×480 pixels. The alternative implementation on the contrary, allows the real-time processing of color frames with sizes up to 2.5MPixels. Both implementations outperform all the similar existing systems.

The above characteristics of the proposed implementations, allows the system to have many potential applications. Such applications are consumer electronics (e.g., digital cameras, mobile phones, video-call systems, and video surveillance systems), robotics (machine vision, assembly lines), driver's assistance (automotive), aerial/ satellite photography and medical imaging.

References

1. Battiato, S., Castorina, A., Mancuso, M.: High Dynamic Range Imaging for Digital Still Camera: an overview. Journal of Electronic Imaging 12, 459–469 (2003)
2. Jobson, D., Rahman, Z., Woodell, G.: A Multi-scale Retinex for Bridging the Gap between Color Images and the Human Observation of Scenes. IEEE Transactions Image Processing 6, 965–976 (1997)
3. Kimmel, R., Elad, M., Shaked, D., Keshet, R., Sobel, I.: A Variational Framework for Retinex. International Journal of Computer Vision 52, 7–23 (2003)
4. Hines, G., Rahman, Z., Jobson, D., Woodell, G.: DSP Implementation of the Retinex Image Enhancement Algorithm. In: Proceedings of the SPIE 5438: Visual Information Processing XIII, pp. 13–24 (2004)
5. Hines, G., Rahman, Z., Jobson, D., Woodell, G., Harrah, S.: Real-time Enhanced Vision System. In: Proceedings of the SPIE 5802: Enhanced and Synthetic Vision, pp. 127–134 (2005)
6. Seponara, S., Fanucci, L., Marsi, S., Ramponi, G.: Algorithmic and Architectural Design for Real-time and Power-efficient Retinex Image/Video Processing. Journal of Real-Time Image Processing 1, 267–283 (2007)
7. Vonikakis, V., Andreadis, I., Gasteratos, A.: Fast Centre-surround Contrast Modification. IET Image Processing 2(1), 19–34 (2008)
8. Benedetti, A.: Image Convolution on FPGAs: the Implementation of a Multi-FPGA FIFO structure. EUROMICRO 1, 123–130 (1998)