Journal of VLSI Signal Processing 38, 35–44, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement

ALI M. REZA
Department of Electrical Engineering and Computer Science, University of Wisconsin-Milwaukee, P.O. Box 784, Milwaukee, WI 53201-0784, USA

Received May 3, 2002; Revised October 29, 2002; Accepted May 14, 2003

Abstract. Acquired real-time image sequences, in their original form may not have good viewing quality due to lack of proper lighting or inherent noise. For example, in X-ray imaging, when continuous exposure is used to obtain an image sequence or video, usually low-level exposure is administered until the region of interest is identified. In this case, and many other similar situations, it is desired to improve the image quality in real-time. One particular method of interest, which extensively is used for enhancement of still images, is Contrast Limited Adaptive Histogram Equalization (CLAHE) proposed in [1] and summarized in [2]. This approach is computationally extensive and it is usually used for off-line image enhancement. Because of its performance, hardware implementation of this algorithm for enhancement of real-time image sequences is sought. In this paper, a system level realization of CLAHE is proposed, which is suitable for VLSI or FPGA implementation. The goal for this realization is to minimize the latency without sacrificing precision.

Keywords: VLSI implementation, system realization, histogram equalization, medical imaging, image enhancement, real-time image processing

1. Introduction

Histogram equalization is a relatively simple method for image enhancement. There are many typical textbook examples in which histogram equalization significantly improves the quality of images with poor lighting. For some examples in typical histogram equalization refer to [3] and [4]. The standard procedure, in this case, is to re-map grayscales of the image so that the resultant histogram approximates that of the uniform distribution. This procedure is based on the assumption that the image quality is uniform over all areas and one unique grayscale mapping provides similar enhancement for all regions of the image. However, when distributions of grayscales change from one region to another, this assumption is not valid. In this case, an adaptive histogram equalization technique can significantly outperform the standard approach.

The main idea in adaptive histogram equalization is to find the mapping for each pixel based on its local (neighborhood) grayscale distribution. In this method which independently developed in [5, 6], and [7], the contrast enhancement mapping applied to a particular pixel is a function of the intensity values immediately surrounding the pixel. The number of times that this calculation should be repeated is the same as the number of pixels in the image. This rises to an extensive computation requirement, which even with some modification, cannot be utilized for real-time image enhancement.

The method that compromises between global histogram equalization and fully adaptive algorithm is the regional histogram equalization. In this case, the image is divided into a limited number of regions and same histogram equalization technique is applied to pixels in each region. This method, which can be optimized, for real-time processing, has one major flaw. The problem with this approach is that, due to differences between mappings of two neighboring regions, the pixels on two sides of the boarder will be mapped differently and consequently a significant blocking effect will be resulted.

An alternative to the fully adaptive algorithm is to approximate the mapping required for each pixel by choosing a small number of contextual regions within the image and calculating the mapping for a given pixel as a bilinear interpolation of the mappings derived from nearby contextual regions. In other words, the mapping for a pixel is obtained by using a weighted-sum of the mappings of its four nearest regions. The weights are calculated based on the proximity of the pixel to the centers of the four nearest regions. A full description of this method may be found in [7] and [8].

In some cases, when grayscale distribution is highly localized, it might not be desirable to transform very low-contrast images by full histogram equalization. In these cases, the mapping curve may include segments with high slopes, meaning that two very close grayscales might be mapped to significantly different grayscales. This issue is resolved by limiting the contrast that is allowed through histogram equalization. Combination of this contrast limiting approach with the aforementioned adaptive histogram equalization results in what is referred to as Contrast Limited Adaptive Histogram Equalization (CLAHE) discussed in [1] and summarized in [2]. Complete formulation of this approach is briefly reviewed in the next section. However, before starting the next section we shall provide a brief review of some of implementation issues related to adaptive histogram equalization.

Some effort in efficient implementation of the CLAHE algorithm is reported in [9]. In this paper a design of a multiprocessor adaptive histogram equalization machine is reported which is used for implementation of the CLAHE algorithm on still medical images. With this implementation the researchers were able to significantly reduce the processing time with their current technology. For a 512 × 512 image the processing time was reduced from 1–2 hours down to 10 seconds. This processing time was and still is reasonable for processing of still medical images. Another parallel processing method for implementation of adaptive histogram equalization is reported in [10]. In this work a speed-up of 6.5:1 is reported for an 8-node machine and a speed-up of 16.5:1 is reported for a 26-node machine. Both of these methods are expensive and only reasonable for still images. Other approaches in this area are reported in [11] and [12]. These methods are not suitable for implementation of CLAHE algorithm that is accepted in the medical community and are only suitable for the reported application in those works.

2. Contrast Limited Adaptive Histogram Equalization (CLAHE)

Contrast limited adaptive histogram equalization has produced good results on medical images. This method is formulated based on dividing the image to several non-overlapping regions of almost equal sizes. For 512 × 512 images, to achieve good statistical estimation, the number of regions is generally selected to be equal to 64 by equally dividing the image by 8 in each direction. One example of such division is shown in Fig. 1. This partition results in three different groups of regions. One group, which consists only of four regions, is the class of *corner regions* (CR). The second group, which consists of 24 regions, is the class of *boarder regions* (BR). All regions on the image boarder, excluding the corner regions, belong to this class. The last group, which consists of all the remaining 36 regions, is called the class of *inner regions* (IR).

In this approach, first, histogram of each region is calculated. Then, based on a desired limit for

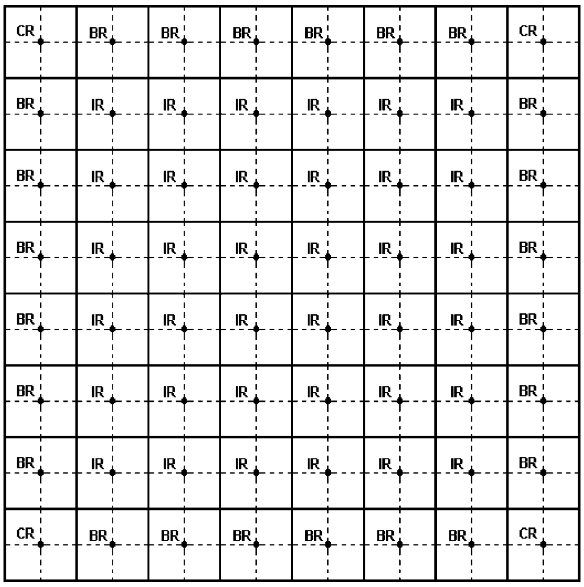


Figure 1. Structure of regions in a 512 × 512 square image that is divided to 64 equal size square regions.

contrast expansion, a clip limit for clipping histograms is obtained. Next, each histogram is redistributed in such a way that its height does not go beyond the clip limit. Finally, cumulative distribution functions, CDF [4], of the resultant contrast limited histograms are determined for grayscale mapping. In the CLAHE technique, pixels are mapped by linearly combining the results from the mappings of the four nearest regions. Formulation of this approach for regions in IR group is straightforward. However, for regions in CR and BR groups this formulation requires some special consideration.

2.1. *Calculation of the Mapping Function*

Calculation of histogram for each region is straightforward. In this case, for each grayscale, number of pixels with that grayscale in the region is counted. Collection of these counts for all grayscales is referred to as histogram of that region. This function is in general a rough estimate of the grayscale density function. Histogram equalization is obtained by using an estimate of the CDF. If numbers of pixels and grayscales, in each region, are respectively M and N, and if $h_{i,j}(n)$, for $n = 0, 1, 2, \dots, N-1$, is the histogram of (i, j) region, then an estimate of the corresponding CDF, properly scaled by $(N - 1)$ for grayscale mapping, is

$$f_{i,j}(n) = \frac{(N-1)}{M} \cdot \sum_{k=0}^{n} h_{i,j}(k); \qquad n = 1, 2, 3 \dots, N-1 \tag{1}$$

This function can be used to convert the given grayscale density function, approximately, to a uniform density function. This procedure is referred to as histogram equalization. The problem with this approach is that the region contrast is increased to its maximum. In order to limit the contrast to a desired level, the maximum slope of (1) is limited to a desired maximum slope. One approach in limiting the maximum slope is to use a clip limit β to clip all histograms. This clip limit can be related to what is referred to as clip factor, α in percent, as follows.

$$\beta = \frac{M}{N}\left(1 + \frac{\alpha}{100}(s_{\max} - 1)\right) \tag{2}$$

In this case, for clip factor of zero percent, $\alpha = 0$, the clip limit becomes exactly equal to (M/N), which results into an identity mapping by evenly distributing all regional pixels into all possible grayscales. No change in pixel values will occur in this case. The maximum clip limit, achieved for $\alpha = 100$, will go to the maximum of $(s_{\max} \cdot M/N)$. This means, the maximum allowable slope is $s_{\max}$. Normally $s_{\max}$ is set to four for still X-ray images. However, for any other application, it is recommended to obtain a good choice for $s_{\max}$ by experiment. As clip factor α is changing between zero to hundred, the maximum slope, in each mapping, is changing between one to $s_{\max}$.

Modification of the original histogram, based on the desired limit in change of image contrast, proceeds by limiting the maximum number of counts, for each grayscale, to β. The extra counts, beyond this limit, are uniformly distributed among the grayscales with counts less than the clip limit. The distribution is carried out in such a way that at no time, the count for any grayscale goes beyond β. This simple redistribution of counts, for each histogram may require several iterations. In general, the number of iterations may increase as percent clip factor decreases. This iterative redistribution algorithm is explained in Algorithm 1. For each region, the grayscale mapping is obtained by using (1) on its modified histogram.

2.2. *Combination of Mapping Functions*

For regions in IR group each quadrant of the region is mapped based on the mappings of its four nearest neighboring regions. For example, with reference to Fig. 2(a), a given pixel in quadrant 1 of (i, j) region is mapped based on its vertical and horizontal distances from centers of (i, j), $(i, j-1)$, $(i-1, j)$ and $(i-1, j-1)$ regions. These distances are shown in Fig. 2(b). If the mapping function for pixels in (i, j) region is represented by $f_{i,j}(\cdot)$, then the new value of pixel p in the first quadrant of (i, j) region, based on the specified distances is

$$\begin{aligned} p_{\text{new}} = {} & \frac{s}{r+s}\left(\frac{y}{x+y} f_{i-1,j-1}(p_{\text{old}}) + \frac{x}{x+y} f_{i,j-1}(p_{\text{old}})\right) \\ & + \frac{r}{r+s}\left(\frac{y}{x+y} f_{i-1,j}(p_{\text{old}}) + \frac{x}{x+y} f_{i,j}(p_{\text{old}})\right) \end{aligned} \tag{3}$$

```
Excess = 0
For $n = 0,1,2,\cdots,N-1$
    If $h(n) > \beta$, Then
        Excess $\leftarrow$ Excess + $h(n) - \beta$
        $h(n) \leftarrow \beta$
    End If
End For
$m$ = Excess/$N$
For $n = 0,1,2,\cdots,N-1$
    If $h(n) < \beta - m$, Then
        $h(n) \leftarrow h(n) + m$
        Excess $\leftarrow$ Excess $- m$
    Else If $h(n) < \beta$, Then
        Excess $\leftarrow$ Excess $- \beta + h(n)$
        $h(n) \leftarrow \beta$
    End If
End For
While Excess > 0
        For $n = 0,1,2,\cdots,N-1$
            If Excess > 0, Then
                If $h(n) < \beta$, Then
                    $h(n) \leftarrow h(n)+1$
                    Excess $\leftarrow$ Excess $- 1$
                End If
            End If
        End For
End While
```

Algorithm 1. Redistribution of histogram for contrast limited histogram equalization.

Similar mappings can be derived for pixels in other quadrants of (i, j) region.

For regions in BR group, the neighborhood structure is different. One such case is shown in Fig. 3(a). In this case, the neighborhood structure for pixels in quadrants 1 or 3 is the same as that of regions in the IR group. However, the neighborhood structure for pixels in quadrant 2 or 4 is different. For example, the structure for quadrant 2 is shown in Fig. 3(b). In this case, the new grayscale for pixel p is obtained by

$$p_{\text{new}} = \frac{s}{r+s} f_{i,j-1}(p_{\text{old}}) + \frac{r}{r+s} f_{i,j}(p_{\text{old}}) \tag{4}$$

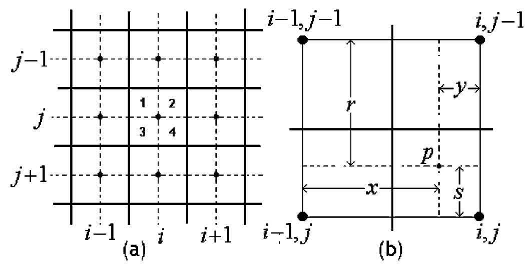


Figure 2. (a) A given IR region with all of its neighboring regions. (b) Pixel p from quadrant 1 of (i, j) region and its relation with respect to the centers of its four nearest regions.

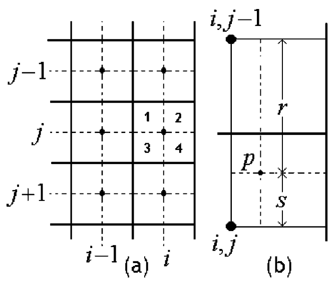


Figure 3. (a) A given BR region with all of its neighboring regions. (b) Pixel p from quadrant 2 of (i, j) region and its relations with respect to the centers of its two nearest regions.

Similar mapping can be derived for pixels in quadrant 4 of (i, j) region. Note that the (i, j) region shown in Fig. 3(a) is from right hand side boarder of the image. All right hand side regions in BR group are similar. The regions in the BR group that are from left hand side, top, and bottom boarders behave similarly. In those cases, the two quadrants, which require mapping similar to (4), are not the same as those given in Fig. 3(a).

For regions in CR group, different quadrants have different characteristics. One typical region in that group, the top-left corner, is depicted in Fig. 4. With reference to the discussions on IR and BR groups, quadrant 4 has neighborhood structure similar to those of IR regions and quadrants 2 and 3 have neighborhood

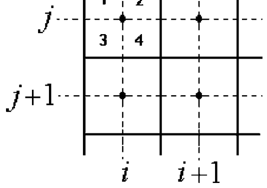


Figure 4. The top-left corner region and its neighborhood structure.

structures similar to the two side quadrants of BR regions. Quadrant 1, in this group, is unique and has no contact with other regions. Mapping function for pixels in this quadrant is the same as the regional mapping with no consideration of other regions. In this case, the mapping would be

$$p_{\text{new}} = f_{i,j}(p_{\text{old}}) \tag{5}$$

The other three CR regions behave in a similar way.

In summary, histogram of each region is calculated and then modified by using Algorithm 1 based on the desired clip factor. Grayscale mapping of each region is obtained by applying (1) to the modified histogram of that region. The final image is built by mapping each pixel based on the Contrast Limited Adaptive Histogram Equalized mappings of regions whose centers are in the four nearest, two nearest, or one nearest neighbor(s) of that pixel. The way that these mappings are combined depends on the vertical and horizontal distances of the pixel from those nearest centers. The equations for these combinations are similar or the same as Eqs. (3), (4), and (5).

3. A Hardware Realization System

Application of CLAHE algorithm on a single video frame demands extensive computation time. Approximate computation time for a n-bit $N \times N$ image frame, which is divided into $\lceil \frac{N}{64} \rceil \times \lceil \frac{N}{64} \rceil$ regions of size 64×64, on a general-purpose computer is estimated at about $(2^{n-9} + 2^7) \cdot N^2$ processing clock cycles. For a 16-bit 512×512 image this amounts to 64 Mega cycles. For an 8-bit 512×512 this value is reduced to its lower limit of 32 Mega cycles. If we assume that a dedicated special purpose computer is solely working on CLAHE algorithm for real-time processing, the processor needs to work at least at the rate of 2 GHz. Although this may appear that it is not out of reach with today's technology but it is pushing the limit and will be very costly.

Our objective in this hardware implementation is to use available low cost FPGA technology (or VLSI implementation for high volume production) for application of CLAHE algorithm on real-time image sequences (videos). The application in mind is a real-time medical X-ray imaging system, which uses continuous low-level exposure video until the region of interest is identified for the final high-level exposure still imaging.

The proposed architecture will meet our objectives due to the fact that we can use combination of parallel processing along with pipelining to achieve significant efficiency in calculation time. Also application of look-up tables and use of local memory results in efficient operation of computational tasks like division and multiplication. Advantages of the proposed approach over general-purpose processor, multiprocessor, or parallel implementation is that the end product of the FPGA or VLSI design is on a single chip and is suitable for special purpose medical imaging system without significantly affecting the price of the system for this added functionality. In the proposed design it is also desirable to limit the intermediate memory requirement so that they can all be utilized on the same VLSI or FPGA chip. Whenever possible, independent calculations are carried out in parallel and serial calculations are done in a pipelined structure. The objective is to minimize the overall latency for this operation.

In the following presentation, a bottom-up approach is used in the sense that individual processing or computational units are first discussed and then their integration is presented. For simplicity of the discussion and better presentation of the main idea it is assumed that the image is 512×512 and it is divided into 8×8 (64) square regions of size 64×64. For other sizes, similar general approach will be applicable and only implementation details are different. For example for different image sizes the size of regions on the borders (on the right border and bottom border) might be different than 64×64 and also the total number of regions would be different. The boarder regions are allowed to be smaller or larger than 64×64 such that to avoid impractical small size regions. For each region we only use the existing pixels in that region to calculate and properly normalize the corresponding histogram. For the clarity of this presentation, details of the control unit and special treatment of boarder regions with different sizes are not discussed and left to be dealt with during the final hardware implementation. Also many issues are left flexible for the hardware designer to adjust and finalize at final stages of the implementation.

3.1. *Calculation of the Clipped Histogram*

It is assumed that calculations of histograms and their original clipping, for at least eight regions in a row, are computed in parallel. One engine for redistribution of excess counts in the clipped histograms and estimation of the regional grayscale mapping functions, if properly

```
Initialization:
Excess = 0
For $n = 0,1,2,\cdots,N-1,\quad h(n) = 0$
Regional pixel access:
$\ell$=The newly arrived pixel value
If $h(\ell) < \beta$, Then
    $h(\ell) \leftarrow h(\ell)+1$
Else
    Excess $\leftarrow$ Excess + 1
End
```

Algorithm 2. Pseudo codes for Fig. 5.

optimized, might be sufficient for all eight regions in a given row. As pixels from each row of the image are accessed, each engine (dedicated to a given region), simultaneously calculates the regional histogram, limits the bin-counts to β (the given clip limit), and finds the excess counts. Pseudo codes for this operation is shown in Algorithm 2 and block diagram of its hardware realization is shown in Fig. 5.

For each pixel, the histogram and excess counter can be updated in less than six clocks. There will be no need for a buffer if the clock rate of the histogram engine is at least six times faster than the pixel arrival-rate. However, if the clock rate is the same as pixel arrival-rate, there is a need for a buffer, with the size of 64 pixels on each engine. In this case, when the last pixel of each regional row arrives, there is still 53 pixels left for processing. The time for this processing is less than 318 clocks, which is less than the time that it takes for the first pixel of the next regional row, exactly $7 \times 64 = 448$

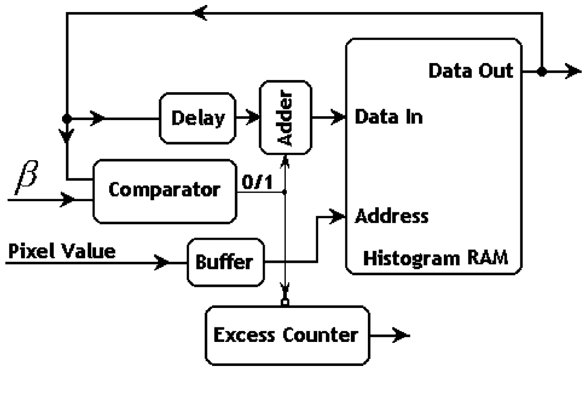


Figure 5. Histogram calculation and original excess counter.

clocks, to arrive. After the last pixel of each region is processed, there are about 130 clocks available to transfer data from histogram RAM's to its corresponding mapping RAM's and reset everything for the next set of histogram calculation. If there is not enough time for data transfer and reset of the histogram RAM's, then it is best to use two sets of histogram RAM's and alternate them for data transfer and histogram calculation. This approach presents no problem in terms of data transfer from histogram RAM's to mapping RAM's in the next stage.

In today's technology, it is possible to have FPGA's with clock rates of up to 160 MHz with enough memory RAM's on the chip, e.g., Xilinx Virtex$^{\text{TM}}$, to design the whole system on a single chip. Therefore, if high-end FPGA's are used, it is feasible to have a clock rate four to six times higher than the pixel arrival-rate. In this case, then buffers in the histogram engines and extra histogram RAM's are not needed. Specific details for this analysis and design should be worked out during implementation time.

3.2. *Calculation of the Mapping Function*

When regional clipped histograms (along with their excess counts) are completed and transferred to their corresponding mapping RAMs, histogram redistribution and grayscale mapping calculation starts. Redistribution operation, in pseudo code, is shown as part of Algorithm 1. In this part, a more complex algorithm is developed whose pseudo code is shown in Algorithm 3 and its corresponding hardware realization is shown in Figs. 6 and 7. The objective in the modified algorithm is to reduce the number of iterations, needed for excess count redistribution. The result is the same as the original redistribution algorithm with no loss of details in the modified routine.

With rough calculation, computation time for redistribution of regional histograms can be estimated. We assume that the maximum number of iterations, required for completing this operation, is three. Based on the assumption that each addition, subtraction, and comparison takes only one clock, each step of the algorithm takes at most seven clocks to complete. An iteration of the redistribution algorithm, for 256 histogram bins, takes at most 1800 clock cycles. This includes initialization of all registers. For three iterations, this number raises to the maximum of 5400 clock cycles. In our simulations, for all practical images, the proposed redistribution algorithm did not take more than

```
Start the loop:
$m = \lfloor \text{Excess}/N \rfloor$
$r = \text{Excess} - m \cdot N$
For $n = 0,1,2,\cdots,N-1$
  If Excess > 0, Then
    If $h(n) < \beta - m$, Then
      If $r > 0$, Then
        $h(n) \leftarrow h(n) + m + 1$
        $r \leftarrow r - 1$
        $\text{Excess} \leftarrow \text{Excess} - m - 1$
      Else
        $h(n) \leftarrow h(n) + m$
        $\text{Excess} \leftarrow \text{Excess} - m$
      End If
    Else If   $h(n) < \beta$, Then
        $t = \beta - h(n)$
        $h(n) = \beta$
        $\text{Excess} \leftarrow \text{Excess} - t$
        $r \leftarrow r + m - t$
    End If
  End If
End For
If Excess > 0, Then
  Go To Start
Else
  End
End If
```

Algorithm 3. Pseudo codes for redistribution of excess counts in the clipped histogram.

three iterations in its operation. In practice, however, it is possible that the algorithm proceeds to further iterations. For practical reasons, therefore, we should limit number of iterations to a fixed value, e.g., three, and terminate the redistribution operation at that point.

The time that it takes the redistribution engine to complete its operation depends on the clock rate. If the clock rate is the same as pixel arrival-rate, then during each histogram redistribution, at most eleven rows have been arrived and processed in the previously discussed eight histogram engines. With proper selection of clock rate, for redistribution engine, one engine is sufficient to redistribute the remaining clipped histograms and be prepared for a new set of clipped-histograms before they arrive. Otherwise, more redistribution engines or a buffering scheme could be utilized.

The resultant histograms should be used, by another engine, to obtain the regional grayscale mappings. This mapping is obtained by calculation of CDF for the corresponding histogram. When, each histogram is ready, grayscale mapping is obtained by simple accumulation and scaling operations. The scaling operation can be set in such a way that it can be carried out by bit shifting. This is possible by approximating $(N-1)/M$ by 2^{-m} for integer m, e.g., for 8-bit images and blocks of 64×64, we get $(2^8 - 1)/2^{12} = 2^{-4} - 2^{-12} \cong 2^{-4}$. Equation (1) represents the mathematical form of this operation. Realization of (1) is shown in Fig. 8.

Calculation of each point of the mapping function takes at most eight clock cycles. The entire function, for 256 grayscales, takes at most 2048 clock cycles to complete. Even if this operation is combined with redistribution operation, with proper clock rate (at least twice

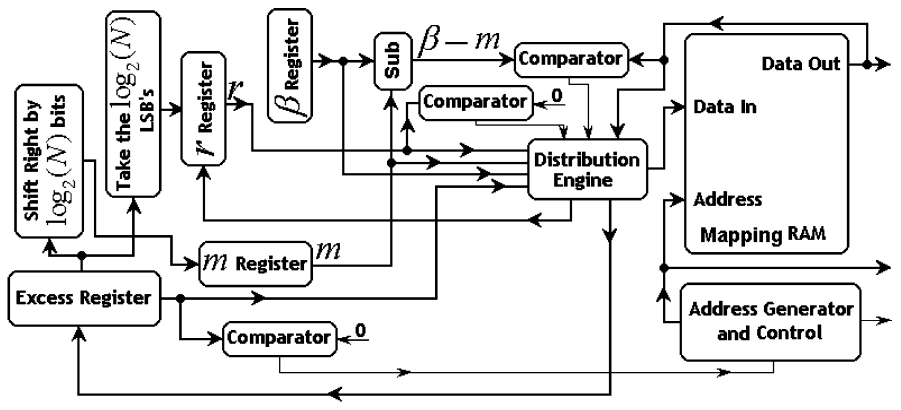


Figure 6. Realization of redistribution routine: register m is updated once after each iteration, register r is updated after update of each histogram bin and is reset once after each iteration.

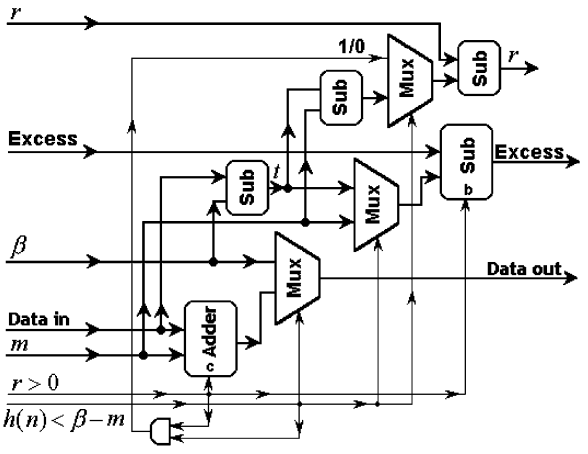


Figure 7. Realization of Distribution Engine used in Fig. 6.

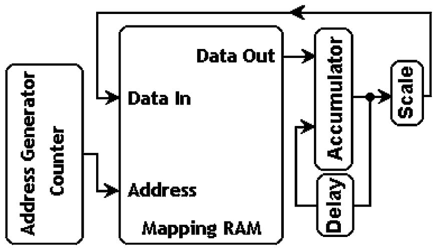


Figure 8. Realization of mapping function calculation.

of the pixel arrival-rate), all eight regional grayscale mappings are completed in time before new set of clipped histograms arrive. For the final mapping of pixels, shown later, there is a need for at least three sets of eight mapping RAM's (24 RAM's each with 256 8-bit words). One of these three sets is always dedicated to redistribution and mapping function calculation. The remaining two sets are used as mapping LUTs to conduct the final mapping of the image. When redistribution and mapping engines complete their operations on a given set, they switch to the next available set, which is not needed as mapping LUTs anymore. Proper timing and clock rates guarantees availability of mapping LUTs and mapping RAM's in their due times. Even when there is a limit in the timing issue, one extra set of mapping RAM's will resolve the problem. Another way to speed up this operation, is to divide the corresponding mapping RAM to 2, 4, or 8 equal blocks and redistribute the excess counts, in parallel, among all bins.

3.3. Mapping Operation of the Image

The next and final operation is to map pixels of the image, according to the proper combination of the regional mappings. For the first 32 rows of an image, only eight regional mappings are needed. However, for the pixels beyond 32nd row, at least sixteen regional mappings are needed. These mappings correspond to two consecutive rows each with eight 64×64 blocks. Not all of these mappings should be ready at once. The time laps between utilization of the first regional mapping in a row and the last regional mapping in the same row, for 512×512 images, is, at most, equal to the arrival time of $7 \times 64 = 448$ pixels. This time is less than the time needed to do the operation shown in Fig. 8 if its clock rate is the same as pixel arrival-rate. In this case, when all operations are conducted at the pixel arrival-rate, there is a need for another extra eight mapping RAM's to work as a buffer for proper pipelining of all operations. Under best circumstances, using a higher clock rate, the minimum required number of mapping RAM's is 24. However, in the worst-case, using a clock rate equal to pixel arrival-rate, total number of required mapping RAMs is 32. For 8-bit images, each RAM is $256 \times$ 8-bit, which rises to 8K Bytes RAM's in the worst-case scenario.

The final pixel mapping is planned based on the assumption that the needed regional mappings are available for the pixel under process. Details of this operation should be pipelined. Realization of this engine depends on the way that six products in (3) are calculated. It is possible to simplify (3) for hardware realization as follows. Let η and μ be defined as

$$\eta = \frac{y}{x+y}, \quad \text{or } 1-\eta = \frac{x}{x+y} \tag{6}$$

and

$$\mu = \frac{s}{r+s}, \quad \text{or } 1-\mu = \frac{r}{r+s} \tag{7}$$

The mapping is simplified by using (6) and (7) in (3), as follows

$$\begin{aligned} p_{\text{new}} = {} & \mu[\eta f_{i-1,j-1}(p_{\text{old}}) + (1-\eta) f_{i,j-1}(p_{\text{old}})] \\ & + (1-\mu)[\eta f_{i-1,j}(p_{\text{old}}) + (1-\eta) f_{i,j}(p_{\text{old}})] \end{aligned} \tag{8}$$

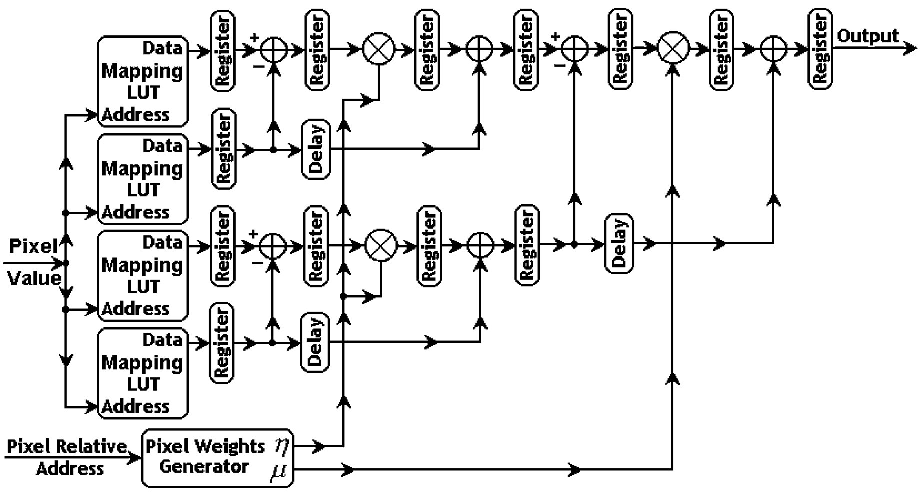


Figure 9. Final grayscale mapping with six multipliers.

This equation can be implemented by using two intermediate parameters as shown in the following equations.

$$\begin{cases} p_1' = \eta \cdot [f_{i-1,j-1}(p_{\text{old}}) - f_{i,j-1}(p_{\text{old}})] + f_{i,j-1}(p_{\text{old}}) \\ p_2' = \eta \cdot [f_{i-1,j}(p_{\text{old}}) - f_{i,j}(p_{\text{old}})] + f_{i,j}(p_{\text{old}}) \\ p_{\text{new}} = \mu \cdot (p_1' - p_2') + p_2' \end{cases} \tag{9}$$

Since the scales in the multipliers, for square regions with size of powers of two, are dyadic numbers (integer numbers divided by powers of two), the multipliers may have a more efficient hardware implementation. A block diagram of this hardware realization for the final mapping, with only three multipliers, is shown in Fig. 9.

If the multipliers are also properly pipelined, with a latency of at most 30 pixels, for each pixel arrival, one pixel mapping is completed. The number of clocks needed to complete each mapping operation depends on the way that the multipliers are implemented. Each multiplication in Fig. 9 is a fixed-point multiplication. Only one engine for this operation is sufficient. In the case of 512 square images, the regions are of size 64 by 64 and the scaling factors in (9) are integers divided by 64.

3.4. Overall Block Diagram

The overall process, including all steps of the CLAHE algorithm, is depicted in Fig. 10. In this block diagram, the flow of the signal as well as the steps and stages of the hardware blocks, with respect to each other, are shown. This block diagram represents the worst-case scenario and includes maximum number of RAM's for proper timing of different stages. The initial image buffer is relatively large and cannot be included in the same chip along with the other parts of the design.

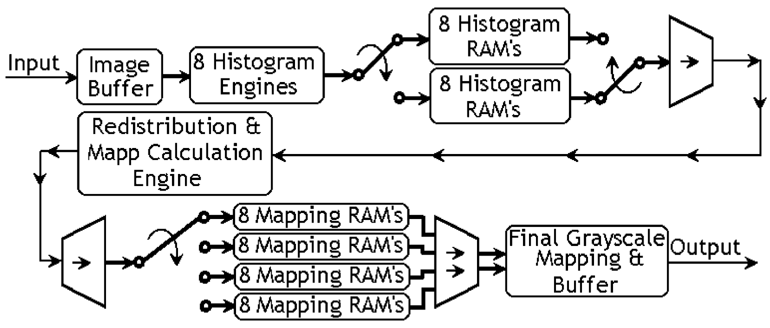


Figure 10. The overall hardware realization block diagram.

This part is at most about half of the one frame, e.g., 128 K Bytes. Other memories can easily be included inside the VLSI chip and is at most about 12 K Bytes.

In this case, the latency would be about half of a frame, which corresponds to about 1/60 seconds in a 30 frames/second image sequence. This realization is therefore feasible for real-time image enhancement.

In terms of different size images, it is possible to use a fixed size 64 × 64 blocks and have special considerations for incomplete regions on the right side and bottom of each frame. The number of histogram and mapping RAM's should be modified according to the image sizes. In this approach, the only concern is on how to complete the regional histograms for incomplete regions. If this part is properly addressed, the remaining operations can proceed as usual with no need to add extra rows and/or columns to complete the incomplete blocks. One approach to complete histograms of incomplete regions is to assume that the missing parts satisfy a uniform distribution and the only required modification for obtaining the correct mapping function would become a modification of scale in (1) as follows.

$$f_{i,j}(n) = \frac{(N-1)}{M'} \cdot \sum_{k=0}^{n} h_{i,j}(k); \quad n = 1, 2, 3, \ldots, N-1 \qquad (10)$$

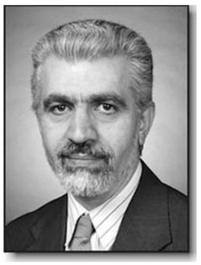
In (10), M' is the actual number of pixels in the incomplete region.

4. Conclusion

In this paper, a particular realization of the contrast limited adaptive histogram equalization algorithm is proposed, which is suitable for VLSI or FPGA implementation. The requirements for this realization are nominal and make it suitable for high-speed VLSI as well as high-end FPGA implementation. The maximum latency encountered in this approach is about half of a frame. The engineering details of this realization are left out for the purpose of generality and flexibility as well as clarity of the presentation.

References

1. S.M. Pizer, E.P. Amburn, J.D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B.M. ter Haar Romeny, J.B. Zimmerman, and K. Zuiderveld, "Adaptive Histogram Equalization and its Variations," *Computer Vision, Graphics and Image Processing*, vol. 39, 1987, pp. 355–368.
2. K. Zuiderveld, "Contrast Limited Adaptive Histogram Equalization," Chapter VIII.5, *Graphics Gems IV*. P.S. Heckbert (Eds.), Cambridge, MA, Academic Press, 1994, pp. 474–485.
3. A.K. Jain, *Fundamental of Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
4. R.C. Gonzalez and R.E. Woods, *Digital Image Processing*. Addison-Wesley Publishing Company, Reading, MA, 1993.
5. D.J. Ketcham, R. Lowe, and W. Weber, "Real-Time Enhancement Techniques," *Seminar on Image Processing*, Hughes Aircraft, 1976, pp. 1–6.
6. R. Hummel, "Image Enhancement by Histogram Transformation," *Computer Vision, Graphics and Image Processing*, vol. 6, 1977, pp. 184–195.
7. S.M. Pizer, "An Automatic Intensity Mapping for the Display of CT Scans and Other Images," in *Proceedings of the VIIth International Meeting on Information Processing in Medical Imaging*, 1983, pp. 276–309.
8. S.M. Pizer, J.B. Zimmerman, and E.V. Staab, "Adaptive Grey Level Assignment in CT Scan Display," *Journal of Computer Assisted Tomography*, vol. 8, 1984, pp. 300–308.
9. S.M. Pizer, R.E. Johnston, J.P. Ericksen, B.C. Yankaskas, and K.E. Muller, "Contrast-Limited Adaptive Histogram Equalization: Speed and Effectiveness," in *Proceedings of the First Conference on Visualization in Biomedical Computing*, 1990, pp. 337–345.
10. C.W. Kurak, Jr., "Adaptive Histogram Equalization: A Parallel Implementation," in *Proceedings of the Fourth Annual IEEE Symposium on Computer-Based Medical Systems*, 1991, pp. 192–199.
11. M. Csapodi and T. Roska, "Adaptive Histogram Equalization with Cellular Neural Networks," in *Proceedings of the 1996 Fourth IEEE International Workshop on Cellular Neural Networks and their Applications*, 1996, pp. 81–86.
12. D. Martinez, "Online Adaptive Histogram Equalization," Neural Networks for Signal Processing VIII, *Proceedings of the 1998 IEEE Signal Processing Society Workshop*, 1998, pp. 531–538.



Ali M. Reza is a professor in the Department of Electrical Engineering and Computer Science at the University of Wisconsin-Milwaukee. His research activities are in the area of Signal and Image Processing. He has conducted research in the multi-sensor signal processing, signal detection, image processing, pattern recognition, and VLSI implementation of DSP algorithms in FPGAs.

Dr. Reza has more than sixty articles that are published as book chapters, research articles in refereed journals and proceedings of numerous international scientific meetings. He has also written numerous internal technical documents based on his research and industrial consulting work.
reza@uwm.edu