Universitat Politècnica de Catalunya

Master’s Thesis

Dynamic Histogram Thresholding for Pixel Classification Algorithms

Author: Norman Tong
Supervisors: Dr. Chunyan Wang
Dr. Jordi Madrenas

July 14, 2015
Resum

Les imatges no sempre són obtingudes en òptimes condicions. Contrast, rang dinàmic o il·luminació poden ser factors que facin no possible la obtenció de tota la informació del món real. Diverses tècniques treballen en aquesta àrea per intentar millorar la qualitat de la informació que podem obtenir de les imatges. La millora del contrast és una de les tècniques més comunes en el pre-processat d’imatges per resoldre aquest tipus de problemes.

El procés més habitual per aconseguir millorar el contrast de les imatges és l’equalització de l’histograma. Utilitzant aquesta equalització sorgeix el problema de que no només el senyal útil és amplificat, el soroll també queda amplificat. Establir un procés de filtrat pas-baix és una manera efectiva per eliminar el soroll generat i obtenir una millora del contrast de gran qualitat amb una quantitat de soroll reduït. A pesar d’això, s’ha d’anar amb compte per no eliminar informació important de la imatge. Per tant, un bon algorisme de classificació de píxels és necessari per eliminar el soroll sense destruir la imatge.

Diferents mètodes de classificació s’han presentat durant els anys. En aquesta tesi, una millora d’un d’aquests mètodes és proposada. El seu objectiu és actuar sobre imatges amb un gradient severament degradat. Una primera classificació dels píxels és feta, basada en la informació estadística de els valors en la escala de grisos de la imatge. Després d’una primera classificació, l’agrupació de píxels és corregida posant un llindar sobre els valors del gradient per obtenir una classificació final. Aquests grups divideixen la imatge en dues regions: homogènia i no homogènia. Un conjunt de màscares son generades a partir d’aquests grups, les quals son utilitzades pels filtres per eliminar el soroll i protegir el senyal útil. Els resultats de les simul·lacions, els quals són subjectius segons l’observació, demostren una millor protecció en els contorns dels objectes a les imatges, cosa que porta a un millor reconeixement d’objectes i canvis en la imatge.

Després de la presentació de l’algorisme, una implementació sobre FPGA és proposada utilitzant codi VHDL. El sistema s’ha dividit en 3 grans blocs, segons les principals funcionalitats dels sistema: equalització de la imatge, classificació dels píxels i filtratge selectiu. Cadascú d’aquests blocs principals també ha estat dividit en blocs més petits amb funcionalitats més específiques, utilitzant la metodologia top-down bottom-up.
Resumen

Las imágenes no siempre se obtienen en condiciones óptimas. Contraste, rango dinámico o iluminación pueden ser factores que hagan no posible la obtención de toda la información del mundo real. Distintas técnicas trabajan en esta área para intentar mejorar la calidad de la información que podemos obtener de las imágenes. La mejora del contraste es una de las técnicas más comunes en el pre-procesado de imágenes para resolver este tipo de problemas.

El proceso más común para conseguir mejorar el contraste de las imágenes es la ecualización del histograma. Utilizando esta ecualización surge el problema de que no sólo la señal útil es amplificada, el ruido también queda amplificado. Establecer un proceso de filtrado paso-bajo es una manera efectiva para eliminar el ruido generado y obtener una mejora del contraste de gran calidad con una cantidad de ruido reducido. A pesar de esto, se debe ir con cuidado para no eliminar información importante de la imagen. Por lo tanto, un buen algoritmo e clasificación de píxeles es necesario para eliminar el ruido sin destruir la imagen.

Diferentes métodos de clasificación se han presentado a través de los años. En esta tesis, una mejora de uno de estos métodos es propuesta. Su objetivo es actuar sobre imágenes con un gradiente severamente degradado. Una primera clasificación de los píxeles es hecha, basada en la información estadística de los valores en la escala de grises de la imagen. Después de una primera clasificación, la agrupación de píxeles es corregida, estableciendo un lindar sobre los valores del gradiente para obtener una clasificación final. Estos grupos dividen la imagen en dos regiones: homogénea y no homogénea. Un conjunto de mascaras son generadas a partir de estos grupos, los cuales son utilizados por los filtros para eliminar el ruido y proteger la señal útil. Los resultados de las simulaciones, los cuales son subjetivos según la observación, demuestran una mejor protección en los contornos de los objetos en las imágenes, cosa que lleva a un mejor reconocimiento de objetos y cambios en la imagen.

Después de la presentación del algoritmo, una implementación sobre FPGA es propuesta utilizando código VHDL. El sistema se ha dividido en 3 grandes bloques, según las principales funcionalidades del sistema: ecualización de la imagen, clasificación de los píxeles y filtrado selectivo. Cada uno de estos bloques principales también ha estado dividido en bloques más pequeños con funcionalidades más específicas, utilizando la metodología top-down bottom-up.
Abstract

Sometimes pictures are not taken in the optimal conditions. Contrast, dynamic range or illumination can be factors which makes a taken capture not have all the information from the real world. Several techniques work in this area, in order to improve the quantity of information we can obtain from images. Contrast enhancement is one of the most common techniques of image pre-processing to solve this kind of problems.

The most common used process to achieve the contrast enhancement is histogram equalization. However, when equalizing an image, not only signal is enhanced but noise do it too. Setting a low pass filtering process is an effective way to remove the generated noise and have high quality contrast enhancement with low noise. Although some caution has to be taken in order to not removing the important information. Therefore, a good pixel classification algorithm is necessary to remove the noise without destroying the necessary signal.

Several classification methods have been presented in the past. In this thesis, an improvement of one of those pixel classification algorithms is proposed. It aims to images with severely degraded gradient signal. A first coarse classification of the pixels is done based on the statistical information of the image gray level values. After the coarse classification the pixel grouping is corrected with gradient thresholding to obtain a fine pixel classification. These pixels divides the image in two regions: edge and flat region. A set of masks based on the groups are generated to have discriminative filters which removes the noise and protects the signal. The simulation results, by subjective observation, demonstrates a better protection of the edges in the image, which leads to a better recognition of objects and changes in the image.

After the algorithm exposition, an FPGA implementation is proposed using VHDL. The system has been divided in 3 big blocks according to the main functionalities of the system: image equalization, pixel classification and discriminative filtering. Each of these main blocks have been also divided in smaller blocks with more specific functionalities, following a top-down bottom-up methodology.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Matlab Algorithm Simulation</td>
<td>54</td>
</tr>
<tr>
<td>A.2</td>
<td>Image to Memory Matlab script</td>
<td>59</td>
</tr>
<tr>
<td>A.3</td>
<td>Memory to Image Matlab Script</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>Hardware Implementation</td>
<td>62</td>
</tr>
<tr>
<td>B.1</td>
<td>Main Block</td>
<td>62</td>
</tr>
<tr>
<td>B.2</td>
<td>CLAHE Block</td>
<td>68</td>
</tr>
<tr>
<td>B.3</td>
<td>Pixel Classification Block</td>
<td>79</td>
</tr>
<tr>
<td>B.4</td>
<td>Filtering Block</td>
<td>92</td>
</tr>
<tr>
<td>B.5</td>
<td>Image Tiling</td>
<td>97</td>
</tr>
<tr>
<td>B.6</td>
<td>Histogram Wrapper</td>
<td>103</td>
</tr>
<tr>
<td>B.7</td>
<td>Histogram</td>
<td>106</td>
</tr>
<tr>
<td>B.8</td>
<td>CLAHE Clipping</td>
<td>109</td>
</tr>
<tr>
<td>B.9</td>
<td>CLAHE Transformation</td>
<td>112</td>
</tr>
<tr>
<td>B.10</td>
<td>Serial to Parallel</td>
<td>117</td>
</tr>
<tr>
<td>B.11</td>
<td>Sobel Kernel</td>
<td>119</td>
</tr>
<tr>
<td>B.12</td>
<td>Valley Detection Wrapper</td>
<td>122</td>
</tr>
<tr>
<td>B.13</td>
<td>Valley Detector</td>
<td>125</td>
</tr>
<tr>
<td>B.14</td>
<td>Sorter</td>
<td>128</td>
</tr>
<tr>
<td>B.15</td>
<td>Threshold Histogram Computation Wrapper</td>
<td>130</td>
</tr>
<tr>
<td>B.16</td>
<td>Parameters Computation</td>
<td>137</td>
</tr>
<tr>
<td>B.17</td>
<td>Threshold Histogram Computation</td>
<td>140</td>
</tr>
<tr>
<td>B.18</td>
<td>Histogram Thresholding Wrapper</td>
<td>142</td>
</tr>
<tr>
<td>B.19</td>
<td>Histogram Thresholding</td>
<td>145</td>
</tr>
<tr>
<td>B.20</td>
<td>Pixel Grouping Wrapper</td>
<td>147</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>B.21 Pixel Grouping</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>B.22 Classification</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>B.23 Comparison and Binarization</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>B.24 Threshold Computation Edge Wrapper</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>B.25 Threshold Computation Flat Wrapper</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>B.26 Threshold Computation Edge</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>B.27 Threshold Computation Flat</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>B.28 Filter Kernel</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>B.29 Histogram Low-Pass Filter</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>B.30 Square root operations package</td>
<td>172</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Subject of work

Contrast enhancement is one of the basic operations in image processing. This operation increases the appearance of light-dark transition, which facilitates the detection of similar color objects.

A commonly used method to achieve contrast enhancement is histogram equalization (HE), which enhances the global contrast of the image. More advanced methods are adaptive histogram equalization (AHE) processes, which are also developed to enhance the contrast of image details. However, these methods have some problems. While signal is being enhanced, noise is also amplified. Several approaches are being investigated in attempt to minimize the amplified noise in the equalization process. There are some modifications of the AHE process by modulating the transformation functions, which are effective in the noise removal, but it severely limits a good contrast enhancement. In order to avoid this limitation, a set of low-pass filters are commonly used after the equalization process to reduce the noise.

Using low-pass filters allows having an equalization process with high-quality images output and a noise removal step at the end. Using low-pass filters is not a perfect method, as the filters do not discriminate between signal and noise, therefore both will be removed. To overcome this situation, filters should be able to discriminate between information and noise pixels. An image segmentation and pixel grouping process should be applied in order to protect important information and erase the noise using discriminative filters.

The work presented in this thesis is focused in having a better pixel classification in order to avoid removal of important information by the filtering process.

1.2 Motivation and objective

In a heavily digitalized world, capturing the moments in pictures have became something common in our lives. These images are used in different areas, like entertainment, security, medicine... In many of these fields, having a good quality capture is important, not because we can contemplate the beauty of it, but because it gives us important information about different aspects.

Due to hardware limitations or environment characteristics, sometimes obtained pictures can have poor quality. In low contrast medical images, some details can be hidden from our eyes, and in night security camera footage, some faces can be
masked by the dark. Algorithms will be used to improve the quality of bad quality images.

The main objective of the work presented in this thesis is to improve the existing algorithms of pixel classification, in order to have a better segmentation depending on the homogeneity of the gray level values. Having the pixels grouped, a set of masks are created to protect the signal from the filtering process. A FPGA implementation of the resulting algorithm is also presented using VHDL as the hardware description language.

1.3 Structure of the thesis

The presented work is based on a previous algorithm which tries to classify the pixels of images, which its signal quality may have been affected by gradient degradation. In order to achieve satisfactory results, the image histograms are analysed to make a better coarse classification. After obtaining better results, an implementation of the algorithm in a FPGA board in VHDL is proposed.

This thesis is structured in 5 different chapters. After the introduction, in chapter 2, a background knowledge and fundamentals about contrast enhancement is provided. A model to improve the existing way of pixel classification is presented in chapter 3. A FPGA hardware implementation of the algorithm is proposed in chapter 4. The results are for the algorithm itself and the hardware implementation are examined in these two chapters. Finally, the work of the thesis is concluded in chapter 5.
2 Background

2.1 Introduction

The real world has a much greater dynamic range than a photographic print does. Information in images can be shaded due to degradation caused by different limitations in illumination conditions and acquisition devices. Different image enhancement are used to make it easier for visual interpretation and understanding of imagery.

The human eye retina has a static contrast ratio of around 100:1. When the eye moves, it re-adjust its exposure chemically and geometrically by regulating the size of the pupil and adjusting the iris. Acquisition devices capture a whole static picture, which makes it impossible to adapt the focus in each of the different areas of the picture.

There are numerous contrast enhancement techniques, such as gray-level transformation based techniques [1] and histogram processing techniques. Histogram Equalization (HE) is one of the most well-known methods for contrast enhancement due to its effectiveness and simplicity. There are various variations of this method such as Adaptive Histogram Equalization (AHE) or Contrast Limited Adaptive Histogram Equalization (CLAHE).

All these methods are able to increase the contrast detail in the images. However, imperfections are also introduced, as these methods cannot differentiate between signal information and noise. This can be an inconvenience, as erroneous signal can be introduced in the pictures complicating the visual interpretation for humans.

In order to overcome this problem generated by this noise amplification some approaches are being investigated. Noise reduction can be done during or after the equalization. The methods applied during the histogram equalization process consist in the modulation of its transformation function in order to make it adaptive to the input signal. In this case, some relevant information could be lost when removing the noise. When the noise reduction is done after the histogram equalization a series of low-pass filters are applied to the resulting signal in order to try to remove the generated noise. These filters need to be discriminative according to the signal and noise density in order the attenuation of the original signal.

2.2 Contrast Enhancement

In order to eliminate the enhanced noise generated by the HE, a three step low-pass filter is added at the output of the equalization process. If the filters are applied to the image, both the noise and the signal would be removed. That is why a set of
masks are generated accordingly to the statistical properties of the image to protect from the filtering the important information and expose the noise to the filter.

![Diagram](image.png)

Figure 1: General block diagram of a Contrast Enhancement algorithm

### 2.2.1 Histogram Equalization

In the Histogram Equalization block there are three methods that can be used: Global HE, Adaptive HE or Contrast Limited Adaptive HE.

\[
p(X_k) = \frac{n^k}{n} \quad (1)
\]

\[
c(x) = \sum_{j=0}^{k} p(X_j) \quad (2)
\]

Global Histogram Equalization \[2\] [3] enhances the global contrast of images, so the intensities are distributed uniformly on the histogram, giving to the image the maximum dynamic range. Having a probability occurrence of gray level \([1]\), where \(k = 0, 1, ..., L - 1\) and \(n^k\) represents the number of times that pixel appears, the cumulative density function \([2]\) would let us define a transform function to change the pixels of the original image \([3]\), having \(X_0 = 0\) and \(X_{L-1} = 255\) for most of the cases.

\[
f(x) = X_0 + (X_{L-1} - X_0) \cdot c(x) \quad (3)
\]

The equalized image can be obtained from \([3]\) by mapping each pixel level \(x\) with its corresponding new level \(f(x)\).

In this process, the dynamic range of the image is stretched to \(L - 1\). This is effective if the original image has a low dynamic range. The effect of Global HE will
be decreased in high dynamic range images. To overcome this limitation, adaptive histogram equalization (AHE) tries to minimize the problem by using different histograms for each pixel in the image. The histogram for each pixel is be computed with the pixels in a close window range.

While computing a histogram for each of the pixels can be computationally costly, the image is usually divided in a limited number of tiles where the histogram is computed. Then, bilinear interpolation is used in order to prevent apparition of boundaries between tiles when applying different transformation functions to different pixels.

![Figure 2: Contrast limitation in AHE and pixel redistribution](image)

Although AHE produces excellent results in the signal enhancement of the image, it also enhance more noise. Contrast Limited AHE helps to solve part of the problem. Uniform regions are normally represented by high peaks in the histogram. When mapping a narrow range of input intensity values, the output intensity values can have a wide range, perhaps over-enhancing noise. Having a maximum value in the histogram will limit the amount of contrast enhancement and thus the enhancement of noise. As shown in Figure 2, after reducing the number of pixels at one location, the clipped pixels are redistributed across the whole histogram. Applying the histogram equalization to the new histogram, the dynamic range is not stretched as much and the enhancement in both signal and noise are limited. Giving a good clip limit value is having the optimal relation between signal and noise enhancement.

### 2.2.2 Low-Pass Filtering

Low pass filters are commonly known for eliminating noise in images. As the equalization process introduces high frequency noise, filtering will help to decrease the noise levels. Depending on the design requirements different kind of filters will be applied in order to use some of their characteristics.
Mean Filter  Mean filters are simple and easy to implement. It reduces the amount of intensity variation between one pixel and the next one.

\[ \mu = \frac{1}{N} \sum p_i \]  \hspace{1cm} (4)

The filter replace each pixel value in an image with the mean value of its neighbours. A \(N \times N\) kernel is created, which is then convolved with the image. This has the effect of eliminating pixel values which are not representative of their surroundings. With larger kernels we can get more severe smoothing.

![3x3 mean kernel](image)

**Figure 3**: 3x3 mean kernel

Median Filter  Median filters also reduce noise in images. However, under certain conditions, it preserves edges while removing noise better than the mean filter.

The standard median filter uses a square window with variable size. The center pixel in the window is the one to be de-noised. The pixels in the window are listed in order and then the middle one (median one) is taken to replace the original pixel value. Median filter removes noise more aggressively but also signal. If bigger windows are applied, better results are expected for homogeneous regions.

Gaussian Filter  Gaussian filters are similar to mean filters. It also de-noises the image with a kernel computed with different values. The values from the kernel are the result of applying the Gaussian distribution.

\[ G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \]  \hspace{1cm} (5)

The Gaussian filter outputs a weighted average of the neighbourhood pixels, determined by the standard deviation of the Gaussian used to compute the kernel, which will make the furthest pixels have less effect than the closest pixels in the current pixel computation. Gaussian function’s Fourier transform is also single, it does shows oscillations. The shape of the frequency response curve of a Gaussian function is a Gaussian itself, which can help to get rid of high frequency signals pollutions and retain most useful signals. When compared with the mean filter, oscillations in the filter appears in its frequency response.
2.2.3 Pixel Classification and Mask Generation

As the histogram equalization introduces noise into the image, it should be removed minimizing the signal loss. Masks are generated to protect signal from the low-pass filters influence. To distinguish between noise and signal is a difficult process. The main important features in images are edges, so an algorithm should be generated to differentiate edges from noise. If edges are extracted after the image have been equalized, noise is too enhanced and edge detectors cannot differentiate signal and noise. In the other hand, trying to detect edges in bad quality images is also a challenge, but it will not mistakenly detect noise as edges.

Image gradient is commonly used in edge detection. Images with gradient values are created, which its position contains the change in intensity between pixels. Large gradient values are categorized as edges while lower values are flat areas. Determining the threshold between this two categories, when a gradient value stops being a edge to become a flat area is a challenge. A good threshold will protect the signal in the image and expose all the flat areas to be filtered in order to reduce the noise.

As this is a complex process, a set of masks is normally created, in order to have a different filters with different protecting/exposure levels. The first mask only protects the most probable edges, while the next masks keeps protecting more and more until the last mask which only exposes the most probable background pixels. This way, the noise in the background is well removed thanks to filtering it with different filters, but the edges are always protected from the filtering process.
2.3 Relevant previous work

2.3.1 Pixel grouping by histogram thresholding

Several pixel classification methods are being studied in order to improve the contrast enhancement performance. In particular, this work is focused in a classification algorithm with pixel grouping by histogram thresholding. This method uses the image and gradient histogram information to determine the gradient threshold value which delimits each of the pixel groups.

Figure 5: Block diagram of the classification algorithm with pixel grouping by histogram thresholding

Having the distribution of the gradient image, the pixels which have a gradient value in the left position in the histogram have a smaller gradient value, so are more likely to be homogeneous regions, while the pixels with gradient values in the right position in the histogram are probably edges. Based in this fact, a set of thresholds are computed to group the pixels in edge or flat region. A low threshold \( T_h_L \) is found at the point where the gradient histogram has the maximum decreasing slope, and a high threshold \( T_h_H \) is set when the gradient distribution tends to approach a constant.

\[
\frac{d^2H(G_r)}{dG_r^2} = 0 \tag{6}
\]

\[
\left| \frac{dH(G_r)}{dG_r} \right| \approx 0 \tag{7}
\]

Gradient values above the previous mentioned thresholds have high probability in being edges, whereas values below the thresholds are probably flat regions. Although probabilistically is correct, in images with severely degraded gradient, not all the
pixels are correctly classified, as some noise peaks can have high gradient values and some edges can have low ones. To get a more accurate result, the image histogram is used, which will help to get a pre-classification of the pixels.

Taking an image gray level distribution, we can state that there is a major concentration of homogeneous pixels in higher bins than in lower bins. Having the same amount of homogeneous and non-homogeneous pixels, the first group is concentrated in a small range of gray levels, which generates bins with high concentration of pixels, while the second group is dispersed in a more variety of gray levels, resulting in more bins with lower values. Therefore, low bins will have a great number of edge pixels and flat pixels will be mostly in high bins. In this way, two subgroups of pixels are created, one with higher bins (flat group) and another one with lower bins (edge group).

A gradient distribution is obtained from both subgroups separately and the thresholds are computed as previously mentioned, one lower and another higher. The higher threshold is more restrictive with edges, a higher value is required to be an edge. In the other hand, the lower threshold is more restrictive with the flat area, a lower gradient value is needed to be in the flat region. As the higher bins are mostly composed by homogeneous pixels, the highest threshold $Th_H$ will be used, which will restrict better the few edge pixels in the flat group. On the contrary, the lower bins are mostly formed by edge pixels, so the lower threshold $Th_L$ will be used to restrict better the fewer homogeneous pixels. For saving processing time, only one of the thresholds can be computed for each of the groups and not both of them.

A unique mask is obtained using this classification. As commented previously, generally a set of three masks are used in order to have a multi-step filtering process. Having a set of thresholds $(T_{G1}, T_{G2}, T_{G3})$ to distinguish between high and low pixels.
in the image gray level distribution will create different masks with different tolerance in the classification towards edge and flat pixels.

2.3.2 Weakness

Images can differ in many different ways, each one with a different statistical information. Brightness or contrast conditions, having the picture taken during sunset or at noon, specifications of the camera used to capture the image... Having a fixed threshold value for all of them can generate a misclassification of pixels.

Taking the example of Figure 8, the same image can have different histograms depending on the conditions in the capturing moment. The first picture could be taken on a bright and sunny day, while the second one could be taken in a cloudy and darker day. Even if the illumination conditions are different, from the point of view of object presence, both images are identical, since the light conditions have been altered with an image processing software. As both images are the same, the number of edge and flat pixels should should not change between images, but the histogram does, since the number of bright and dark pixels have been altered. In this case, if the threshold used for both images is the same, the number of pixels assign to each group will be different.

Therefore, each image have an optimal threshold which can vary depending on the conditions in the capture moment. A possible solution, which is the one used in the method described previously is to introduce the threshold values manually, but this can require a big amount of time. If the values are introduced manually, a person should try different threshold values until the result is good enough and repeat the process for each single image. This is not a good method, as it requires somebody to spend some time selecting the best threshold and sometimes we can have not only images but videos in which illumination conditions can change from frame to frame.
In order to solve the problem, a set of adaptive gray-level histogram thresholds should be obtained. An analysis of the statistical information of the image should be made, identifying image elements that characterize edge and flat regions in the image. It is important to identify these parameters to produce a proper modeling for the threshold computation.

Once the thresholds are dynamically computed, another limitation appears. The thresholds are only valid for low contrast images. In the case of high dynamic range images
images, gradient degradation can be different for the different gray level ranges. Applying a gradient threshold of the whole image can result in mis-classification of pixels. Again, statistical information can be used to divide the gray level range into a sub-set of ranges and treat the problem locally. In the example in the Figure 5, the histogram can be divided in two ranges, pixels with lower values and higher values. The gradient of the sub-set of pixels will have similar degradation and the gradient threshold will be valid.
3 Description of the Algorithm

3.1 Overview

Varieties of Histogram Equalization based enhancement methods have been reported. However, it is often the case that, while the signal contrast is enhanced, the noise is also amplified. Using low-pass filters to remove the noise generated by the histogram equalization is an effective approach. The critical point is the conflict between noise removal and signal preservation. Therefore, a good classification algorithm should be used to solve this problem. As described previously, the new classification method is based upon a previous algorithm which presented some weaknesses. The work presented in this chapter is focused on developing a solution for this weakness applying dynamic thresholds to the pixel distribution.

3.2 Analysis of the pixel distribution

The method consists in a 2 step classification. The first one is a coarse grouping based on the pixel distribution and the second one is based on the gradient distribution which performs a fine classification. As the coarse classification is the first step, it is important to make a good grouping. This first classification consists in applying a threshold to the image histogram. These thresholds should be different depending on the type of input image.

In any picture we can distinguish to regions, homogeneous and non-homogeneous. The homogeneous regions have big extensions of pixels with similar value, we will refer to them as the flat regions. Meanwhile, the non-homogeneous or edge regions have a great diversity of pixels. Having big extensions of similar pixels results in having a small amount of histogram bins with really high values, while non-homogeneous regions contributes with really low value in a big number of bins scattered in all the histogram range. When computing the histogram of the whole image, the histograms of both regions will be superposed resulting in one histogram with high value bins composed mainly by flat pixels and low value bins composed by edge pixels.

Both edge and flat pixels are mixed in the histogram bins, but the higher bins have a larger proportion of flat pixels and the lower bins have more edge pixels. In order to obtain a first classification, two groups are formed: high bins, mostly with flat pixels, and low bins with edge pixels. To differentiate between bins from both groups, a threshold is applied. Selecting the optimal threshold is a hard task, as bins with really high values are easily classified as high bins and bins with low values as low bins. The problem appears for the bins with middle range values. Having a higher or lower threshold in the middle range can be decisive to obtain a correct
3.2.1 Histogram Threshold

As said before, a threshold is used to differentiate between high and low bin groups. Selecting the correct threshold is important, as a set of thresholds can be optimal for images with similar statistical characteristics but not work as well when its statistical information differs.

Gray level histograms can have various forms, but it is generally considered that higher values correspond to homogeneous areas while lower values are probably corresponding to edges.

It is important to be able to differentiate correctly between the edge pixels (low bins) and the flat pixels (high bins). If the threshold is not applied correctly, it can result in having an over-filtered or too noisy image. In the Figure 11, it can be observed how the image can be too noisy if the threshold selected is too high. All the background and the brushes area in the image still have plenty of noise which has not been removed. By the other hand, in the Figure 12, the image is over-filtered. A threshold which is too low is applied to the histogram image and some of the signal is removed from the image in the filtering process. It can be observed how the table looses sharpness and is too smooth and the couches border are over-smoothed.

For low contrast images, the gray level histogram normally is formed by one peak, which can have different widths. Having a narrow peak normally indicates having a decreased image contrast, as opposition as having a wider peak, which it normally indicates that the image has a higher contrast.
When the contrast in the image is lower, most of the pixels gray levels are concentrated near a central bin in which the maximum of the histogram is. These high bins have more density of flat pixels. But, as most of the pixels have really similar gray levels, the edges are really degraded in the area. If an image has degraded edges pixels, which move to higher bins, in order to catch those pixels, the threshold should be arisen. For wider peaks, the distribution of pixels in the gray level scale is better distributed, resulting in less degraded edges and a lower threshold needed.
3.2.2 Statistical Information

Having a wide variety of images, statistical information can give us information about them. Several statistical parameters have been analysed and considered in order to compute a good threshold for the different kind of images. For the work presented, different parameters from 3 sources have been analysed: the image histogram, the image itself and the gradient histogram.

\( H_{\text{median}} \) - It is the median value of the image gray level histogram. To find this median value, the values of the bins on the histogram are placed in order. The bin positioned in the middle of the ordered list is the one taken as \( H_{\text{median}} \). The gray level corresponding to the selected bin is the median gray level. When computing the median, the value is not distorted by really high or low bins. This will be handful to get a statistical information not dependent of the extreme values.

\( H_{\text{mean}} \) - It is the mean value of the image histogram. The mean is obtained with the equation (8), where \( N \) is the number of bins (number of gray-level values) and \( h_i \) is the value of the \( i \) bin in the histogram. Unlike the median value, the mean is affected by extreme values, like really high peaks in the histograms.

\[
\mu = \frac{1}{N + 1} \sum_{i=0}^{N} h_i
\]

(8)

\( H_{\text{max}} \) - It is the maximum value of the gray level distribution. This value can be computed as the number of repetitions of the gray level value that appears most often in the image. Normally the histogram of the image is already pre-computed, so we will refer to it as \( \text{max}(h_i) \).

\( H_{\text{min}} \) - It is the minimum value of the image histogram. In the presented work, this value is not strictly the minimum value but a variation of it. If the histogram values are ordered in descending position, the minimum value will be the one in a certain position from the bottom ordered bins. The value of this position is the 20% of the length of the histogram. Having this method of computing the minimum value stands for the reason of trying to differentiate the histogram with high value tails from the ones with low value tails.

\( H[GL_{\text{median}}] \) - It is the histogram value at the image median position. In this case, the median is computed over the image itself and not the histogram. When the image median value is obtained, it is mapped with the corresponding histogram bin and \( H[GL_{\text{median}}] \) is obtained. This value divides the histogram in two halves, each half with the same amount of pixels. That means that the sum of the bins in the left half and the sum in the right half is almost the same \( \sum_{i=0}^{GL_{\text{median}}} h_i \approx \sum_{i=GL_{\text{median}}}^{N} h_i \). This equation has not an equal symbol because some pixels in the \( GL_{\text{median}} \) bin should be in the left group and some others in the right one. This value gives
Figure 13: Left histogram has a lower $H_{min}$ minimum value than the right histogram. If the strictly minimum value is taken, both histograms would have approximately the same value for the parameter.

information about symmetry in the histogram. When $H[GL_{median}] \approx H_{max}$, the histogram will present high symmetry.

$H_{grad_{mean}}$ and $H_{grad_{max}}$ represents the mean and the maximum value of the gradient histogram respectively. These values have the same properties as image histogram mean and maximum value, but being obtained from the gradient histogram provides different information which will be explained below.

Although the previous parameters give statistical information about the image, having them combined can give us information in a higher level that we can understand and use.

$H_{mean} \over H_{max}$ - This parameter provides information about the peak width. Considering a histogram image with only one peak, when this peak is really narrow, only few bins have high values, while most of them have low values. Furthermore, when there are few high bins, the difference between the higher and lower ones become greater. This will let to a low $H_{mean}$ and high $H_{max}$ values. When the peak is more wide, we have the opposite effect in the previous parameters. Combined, they provide a better precision in the peak width information.

$H_{median} \over H_{mean}$ - This fraction also gives information about the peak width in a different scale. Because the mean is influenced by extreme values, when the peak is narrow, the values of the bins in it will probably be high, which will influence the mean value to increase, while the median value will stay as a low value. This will result as a low $H_{median} \over H_{mean}$ value. Having a wider peak will result in having a medium value more similar to the mean, so the parameter will have a higher value.

$H[GL_{median}] \over H_{max}$ - As commented previously, this parameter gives symmetric information about the histogram. If $H[GL_{median}] \approx H_{max}$ the histogram will present high symmetry. In this case, the main peak will have, approximately, the same quantity
of pixels on the left and right of it in the histogram shape.

\[
\frac{H_{\text{grad mean}}}{H_{\text{grad max}}} - \text{Gradient is normally related with edges. When an image has a large amount of edges, this values are scattered amongst the histogram, resulting in a bigger relation between } H_{\text{grad mean}} \text{ and } H_{\text{grad max}}, \text{ as the maximum decreases faster than the mean value. This relation provides the amount of edges in the image. As the parameter does not give an exact amount of edges, just a coarse statistical approximation, the gradient degradation does not produce a big distortion in this parameter.}
\]

Most of the commented parameters will be useful to create a model that computes a good threshold value for the histograms different images can have.

### 3.2.3 High Dynamic Range

All the parameters presented in the previous chapter give a valid information for low contrast images histograms, in which their shape consist in one peak formed with high value bins and a tail with low values, but not all the images have this type of histograms. Other type of images that can be degraded are the ones with a high dynamic range.

![Figure 14: High dynamic range image histogram](image)

Acquiring high dynamic range images by a medium range camera can cause gradient compression, due to non-linear transformation functions of the acquisition devices. It is often the case that the compression in the higher ranges is very different from the one in the lower ones. Because of this difference in the compression in the different ranges of the histogram, the high dynamic problem will be divided in a set of ranges to work, with the same characteristics than the low contrast histograms.
The high dynamic histogram can be divided in a set of ranges. For example, we could have 3 groups of bins with these ranges \([0, T_1], [T_1, T_2]\) and \([T_2, 255]\), where in each of them there is a peak. Each of the peaks in these ranges can be counted as a low contrast image and apply the same model for them independently in order to classify the pixels in the edge or flat group.

In the example from Figure 14, the histogram can be divided into 2 sub-histograms with its separation point around the bin number 170. Each of the ranges \([0, 170]\) and \([171, 255]\) are considered as separate and independent histograms in order to compute and apply the threshold. A set of thresholds will be computed for each of the ranges, which will be applied to divide the pixels into the edge or flat groups in each of the ranges separately. Once the pixels in each range have been classified into these 2 groups, they are merged into a single edge and a single flat groups.

### 3.3 Model

#### 3.3.1 Threshold computation

As commented previously, a set of 3 thresholds should be obtained, one for each of the masks of the multi-step low-pass filtering process. The first mask will be focused in protecting as much as possible the edges in the images, while the last mask will expose the most flat regions. The middle mask will be a point in between.

The first filter will try to eliminate noise from most of the areas in the picture. The mask for this filter will have the least protected areas, so it is important to be as much accurate as possible when detecting the edge areas in order to keep the information. The edge area corresponds to lower bins in the image histogram, so this first threshold \(R_1\) will be the lowest and should be accurate discerning between edge and not edge bins.

Having the same image, if degraded in a way that the contrast is reduced, plenty of edge bins will be merged towards the maximum value of the histogram, resulting in higher edge bins and a higher threshold needed to capture these edge pixels. \(R_1 = \frac{H_{\text{mean}}}{H_{\text{max}}}\) and \(R_2 = \frac{H_{\text{median}}}{H_{\text{mean}}}\) are the parameters in charge of compensating this fact.

If there is a good preservation of the edges in the first filter, these edges will be preserved in the other filters, as this one is the one that exposes more areas to the filters. It is important to detect as more edge pixels in this step. The parameter \(R_3 = \frac{H_{\text{grad \_mean}}}{H_{\text{grad \_mean}}}\) is the parameter in charge to evaluate the general quantity of non-homogeneous pixels in the image.
\[ T_{\text{low}} = \left( \frac{1}{1 + 2R_1} \frac{1}{1 + 2R_2} + R_3 \right) \frac{H_{\text{max}} - H_{\text{min}}}{3} + H_{\text{min}} \]  

(9)

In the equation (9), \( H_{\text{min}} \) is a correction parameter. In most of the cases, low contrast images present histograms with a peak and tails where the lowest values are approximately zero. In these cases, \( H_{\text{min}} \) will not have any influence in the threshold computation. There are some exceptions to this general rule, in which the histogram can have a tail with higher values. If that is the case, not using this minimum value would result in having a low threshold with none or too few bins below it. Therefore, this correction parameter must be included in the equation of the first mask.

The latest filter is the one in charge of filtering only the most homogeneous areas. Its mask will try to protect as much area as possible, only exposing to the filter the pixels that are in an area of high probability of being homogeneous. In this case, only a few bins will be considered as flat region, so the latest threshold (10) will have the highest value.

As previously presented in the lowest threshold, the highest one, also changes depending on the histogram statistics, specially depending on the histogram peak. The parameters \( R_1 = \frac{H_{\text{mean}}}{H_{\text{max}}} \) and \( R_2 = \frac{H_{\text{median}}}{H_{\text{mean}}} \) are also present to modify the threshold value depending on the peak width.

\[ T_{\text{high}} = \left( \left( \frac{1}{1 + \left( \frac{1}{1 + \sqrt{R_1}} \right)^2} \right)^{1/2} + \left( \frac{1}{1 + \left( \frac{1}{1 + \sqrt{R_2}} \right)^2} \right)^{1/2} \right) \frac{H_{\text{max}}}{2} \]  

(10)

After defining the lowest threshold which is accurate with edge pixels and the highest threshold which controls the flat pixels, the middle threshold (11) is a step in between both thresholds. Is accurate with edge and flat pixels with the same amount of precision. This is expressed as the middle point between the lowest and the highest threshold.

\[ T_{\text{mid}} = \frac{T_{\text{low}} + T_{\text{high}}}{2} \]  

(11)

The presented thresholds are only valid for low-contrast images with only one main peak. In order to generalize the threshold computation for high dynamic range images, the histogram is divided in different ranges to have only one peak in each one of the ranges. Doing so, the previous equations can be applied locally to that range. Once the thresholds are computed for each range, the different threshold values are applied to the pixels in the image depending on its range position.
3.3.2 Valley Detection

The used feature to divide the histogram in different regions with one single peak is a valley. The valley point is the lowest bin between to peaks. The proposed computation of the valleys in the histogram has been done thinking in a VHDL implementation.

Before locating the valley, the peaks should be detected. As an histogram can have several peaks, the first step is defining them. In this proposed algorithm, the histogram bins are ordered by its value, the highest 10 bins are taken and the bins between each of them are analysed. To consider the existence of a valley, the bins in between the highest ones should have a value below \(0.1H_{\text{max}}\). If bins with values below these threshold are found, the minimum bin of the range is taken as a valley.

Once the valleys in the histogram has been the detected, this is split in different ranges, in which the thresholds are computed individually for each region, treating the problem as a low contrast image.

3.4 Matlab simulation

Matlab simulation is done to evaluate the effectiveness of the proposed algorithm before implementing it in a FPGA. The performance is evaluated with subjective observations of the images.

The simulation has been performed with a set of parameters. The CLAHE algorithm used in the contrast enhancement block is done using tiles of 8x8 pixels size and clip limit of 0.03 to provide a good contrast enhancement. The gradient has been computed with four SOBEL kernels, detecting the variation in the horizontal, vertical and diagonal directions. The filtering process is composed by a set of 3 Gaussian filters with window size of 5x5 and \(\sigma = 1\) to provide a moderate smoothing.

In Appendix A.1 can be found the Matlab code used to run the simulation.

The first example is a picture with two differentiated areas, one is really dark and the other is brighter. It is possible to distinguish all the elements from the picture in the brighter area, but impossible to see them in the darker part of the image.

From the Figure 15 it can be seen that the equalization process allow us to distinguish parts of the image in the background we could not see with our eyes. Before the enhancement process, we could only see easily 2 objects: the table and the flower. Thanks to the contrast enhancement process, new objects appear in the image. These objects were always there but they were hidden to our eyes. The

\(^1\)In order to see better the results, deactivate the anti-aliasing property in the PDF viewer.

28
couches and the tables in the background appear really clear in the second image after equalizing it. We can also observe there a door and distinguish that the wall is not homogeneous. This is due to how the histogram are. In the original image, most of the pixels are concentrated in the really dark areas.

After the equalization process, the accumulated pixels in that part are stretched which allow us to distinguish more objects in the image. Noise can also be observed in the background, where a stronger enhancement has been made. This noise will be removed, while the irregularities of the table and the flower are kept in the filtering process. In the Figure 17 the filtered image is presented with the previous existing
Figure 17: Left is the enhanced image using the previous existing algorithm. Right is the enhanced image using the improvements presented in this chapter.

Figure 18: Left is the original image. Right is the image after applying CLAHE.

pixel classification algorithm and the improved version. It can be seen how in the improved version the table is sharper while the contours of the flower are more precise, erasing better the noise in the background but next to the border of the flower.

The second example used is a standard medical image, an x-ray picture. Due to the limitation of the acquisition devices, these medical images can present really
Figure 19: Left is the histogram of the original image. Right is the histogram of the enhanced image.

Figure 20: Left is the enhanced image using the previous existing algorithm. Right is the enhanced image using the improvements presented in this chapter.

low contrast, which can generate loss of information.

The Figure 18 shows how after the equalization process the rib and the vertebral spine bones appear with better details, but noise is generated. After the filtering process in the Figure 20 it can be seen how the noise is removed in both images, but some signal is lost in image obtained from the previous existing algorithm. The detail in the vertebral spine is improved using the new algorithm, amongst other details like the differentiation between the ribs and the background or other organs.

The objective of the improved algorithm was to be more critic with the edge areas and be conservative in order to not erase important information of the image. In the previous examples, it can be observed how the new improvement keeps better
the signal in the pictures, which let us see them with better resolution.
4 Hardware Implementation

4.1 Introduction

Once the algorithm has been exposed, a hardware implementation using VHDL description is presented in this chapter.

Implementing the algorithm in general purpose devices can be a quick and flexible solution in the deployment phase. However, this solution can be have some speed issues in the working phase, as this processors are able to do any task but are not optimized to do any of them. Although, using a dedicated hardware implementation can give us a lower degree of flexibility, the velocity in the operations execution and the power needed is lower.

The implementation has been divided in 3 big blocks, having distinct purposes. Each of these big blocks have been also divided into several smaller blocks, which will be explained in this chapter.

4.2 System Structure

The system is divided in 3 main blocks, each one with different functionalities: equalization, pixel classification and filtering blocks.

![Global system divided in equalization, pixel classification and filtering blocks.](image)

Figure 21: Global system divided in equalization, pixel classification and filtering blocks.

Two of the main blocks can work in parallel, while the third and last one has to wait for the previous ones. The equalization and pixel classification blocks are not sharing any kind of signal, so they can work in parallel without having to wait one to another. Once both blocks are finished, the filtering block get the outputs of these two previous main blocks and starts filtering the equalized image discriminating the pixels that are masked.
The global block makes use of some input signals. This signals gives flexibility to the system, as the values can be changed to obtain different outputs or to control different images that can be introduced in the system.

1. Reset \( (rst) \) and enable \( (en) \) are 1 bit signals that control the general state of the system. When the \( rst \) signal is enabled, the system is restarted and all the signals are set to their initialization value. The signal \( en \) controls when the system is working. The system is in a standby mode until this signal is enabled.

2. Number of pixels \( (npixels) \) tells how many memory positions are filled with signal information. As the system can be loaded with a different variety of image, which can have different sizes, the memory can have positions without pixel information. Specifying \( npixels \) will tell the system in which memory positions should look into to extract the information.

3. Image width \( (x\text{ size}) \) and height \( (y\text{ size}) \) express the geometry of the image. In the memories, the pixels are stored in one dimension, as the addresses go from zero to \( npixels \), but two dimension information is required for some operations in the equalization, classification and filtering processes.

4. Clip limit \( (clip\text{ limit}) \) is used to limit the contrast in the CLAHE process, which provides flexibility to the system. Its value can be between 0 and 100, as it express the clipping percentage that is applied to the histogram equalization.

The system only have only output \( end\_flag \) which indicates that the equalization process is finished and all the information is stored in the output memory.

4.2.1 Equalization Block

This part is composed by several sub-blocks with the aim of computing the equalized version of the input image. It is divided in 4 different blocks and a set of small memories to store the histogram of each of the tiles in which the image is divided for the equalization process.

In the CLAHE computation, the image is divided in a set of tiles. For each of these tiles, the histogram is computed. To avoid over-amplification of noise, the histograms are clipped. Once done, the image is transformed with the equalization function. This process is transformed in 4 blocks: image tiling, histogram generation, histogram clipping, equalization.

In order for all the components to work together properly, the CLAHE entity \( clahe\_wrapper \) (\( clahe\_wrapper.vhd \)) describes the following functions:
Figure 22: Block diagram from the equalization process. Some components are reused and the connections have to keep changing.

- Switches the connections between the different components and the memories where the histograms are stored. If several components are connected to the same histogram memory block, collisions would appear.
- Manages which components are working. Some components are able to work in parallel but some others have to wait for the previous ones to end.
- Controls the signals synchronization. The process of changing the histogram memory blocks take different time than other signals which are not obtained from the memories. To get a correct computation of the histogram, all the signals must be synchronized.

The instances of the sub-blocks that can be found in the CLAHE entity are the following:

- **image_tiling**: divides the image in 8x8 tiles and outputs the correspondent address and parameters of the tiles.
- **histogram_wrapper**: computes the histogram from a string of entering values.
- **histo_ram2**: memories where the histogram values are stored. A set of tiles(0-63) of memories are generated, one for each of the 64 tiles.
- **clahe_clipping**: clips the histogram bins that exceed the limit specified in the system input and computes the cumulative distribution function (cdf) which is stored in the same histogram memory where the information is read.
- **clahe_equalization**: transform the original image values for the equalized ones using the equalization function.
The `image_tiling` block divides the image in tiles and starts giving addresses that are used to read the pixels from the original image. Those pixels are used by the `histogram_wrapper` to generate the histograms and store them in the histograms ram. This block will repeat the same process for each one of the tiles. Once the `histogram_wrapper` is finished generating the histogram, the `clahe_clipping` block starts clipping the histogram bins and generating the cdf. The cdf value is stored in the same histogram ram to optimize the memory space. After generating all the cdf values, the `clahe_equalization` block starts transforming the pixels in the order provided by the `image_tiling` block, which is enabled again to provide the addresses to the transformation block.

As each tile works independently, after the histogram of one of the tiles has been generated, the clipping block can start working without having to wait for the histogram process to compute the histogram of all the tiles. A control check is needed, so the clipping process can only start working on the tiles where the histogram process has already finished.

### 4.2.1.1 Image tiler

The `image_tiling` block is described in the file `image_tiling.vhd`.

The block has 2 important functions: compute the address needed to be able to work with tiles and compute parameters that characterize the tile for other blocks computations. A set of 3 processes are described to obtain the desired parameters.

When the component is enabled, the first thing that is computed is the tile size. The image is divided in a 8x8 matrix and the size of tile is obtained. Because the width and height of the image normally are not multiples of 8, there will be a few lost pixels at the end of the x and y axis in the image.

Once the sizes have been computed, the block starts scanning all the addresses for each of the tiles, from left to right and top to bottom. In order to compute the final address, few steps are done. First, the local position of the pixel inside a tile is computed \((x_{pos}, y_{pos})\). When the position exceeds the tile size, the size number is incremented \((x_{tile}, y_{tile})\). This way, the block computes the address where the current tile starts \((12)\) and the final address \((13)\).

\[
addr_{tile} = x_{tile} \cdot x_{tile\_size} + y_{tile} \cdot y_{tile\_size} \cdot x_{size} \quad (12)
\]

\[
addr = addr_{tile} + y_{pos} \cdot x_{size} + x_{pos} \quad (13)
\]

For each of the pixel positions, the adjoining tile numbers and the distances
to their center points are computed. As the equalized pixels are computed using interpolation, these values are needed for the transformation block, which also uses the image tiler.

4.2.1.2 Histogram generation

The histogram_shader block is described in the files histogram_shader_int2.vhd and histogram_shader_int3.vhd. This files have been obtained from [9], with a few corrections.

The block is capable to compute each histogram value in one cycle, only with a few cycles of initial latency when the process starts.

When a pixel is introduced, the block reads the position with the same pixel value from the histogram ram, increments the read value and stores it again in the same position. The most recent incremented value is also kept in the histogram block. If the next pixel has the same value, there kept value is incremented automatically without having to read from the memory. This way, the histogram is computed more efficiently and with less clock cycles.

4.2.1.3 Histogram clipping

The clahe_clipping block is described in the file clahe_clipping.vhd.

The block scans all the positions of the histogram twice. In the first scan, each of the bins are compared with the clip threshold. When the value is higher than the threshold, the exceeded value of that bin is added to the global excess [14] and the new bin value is the clip_limit value [15]. After finishing the first scan, the excess is divided by the number of bins in the histogram, in order to redistribute it. In the second scan, the divided excess is added to each one of the bins of the histogram.

\[
\begin{align*}
\text{excess} &= \begin{cases} 
\text{excess} & \text{if } x[n] \leq \text{clip}\_\text{limit} \\
\text{excess} + x[n] - \text{clip}\_\text{limit} & \text{if } x[n] > \text{clip}\_\text{limit}
\end{cases} \\
y[n] &= \begin{cases} 
x[n] & \text{if } x[n] \leq \text{clip}\_\text{limit} \\
\text{clip}\_\text{limit} & \text{if } x[n] > \text{clip}\_\text{limit}
\end{cases}
\end{align*}
\]

While the redistributed excess is being added to the histogram bins, these are being summed to obtain the cdf [16]. This is the new value that is stored in the histogram ram.
\[ cdf[n] = cdf[n - 1] + y[n] + \frac{\text{excess}}{\text{npixels}} \] (16)

### 4.2.1.4 Histogram transformation

The `clahe_equalization` block is described in the file `clahe_equalization.vhd`.

In this block, the original pixel is transformed to the interpolated equalized value. The pixel value is transformed (17) as if it was in the center of the surrounding tiles. Once these new values are obtained for each of the tiles, the distances to those center points \( (d_1, d_2, d_3, d_4) \) are used to compute the real interpolated value (18).

\[ h[n] = \text{round} \left( \frac{cdf[n] - cdf_{\text{min}}}{\text{npixels}} \cdot 255 \right) \] (17)

\[ y[n] = \begin{cases} 
  h[n] & \text{if no interpolation} \\
  \frac{h_1[n] \cdot d_1 + h_2[n] \cdot d_2}{h_1[n] \cdot d_1 + h_2[n] \cdot d_2 + h_3[n] \cdot d_3 + h_4[n] \cdot d_4} & \text{if linear interpolation} \\
  \frac{h_1[n] \cdot d_1 + h_2[n] \cdot d_2 + h_3[n] \cdot d_3 + h_4[n] \cdot d_4}{d_1 + d_2 + d_3 + d_4} & \text{if bilinear interpolation}
\end{cases} \] (18)

The interpolation process can be different depending on the position of the pixel. 3 different types of tiles can be considered: central, side and corner tiles.

The central tiles have only one type of interpolation, which is bilinear interpolation. All the pixels in those tiles are surrounded by other tiles in all their sides. In this case, bilinear interpolation is used to compute the final value.

The side tiles have 2 types of interpolation. Linear interpolation is used for the pixels in the side part of the tile, while bilinear interpolation is used in the inner

![Figure 23: Tile-based interpolation for the efficient computation of adaptive histogram equalization.](image)

The side tiles have 2 types of interpolation. Linear interpolation is used for the pixels in the side part of the tile, while bilinear interpolation is used in the inner
part of the tile. The inner part is surrounded by tiles in all their sides, but the side part only have surrounding tiles in one direction.

Finally, there are 3 ways of computing the final value in the corner tiles. Similar to the side tiles, there are 2 types of interpolation for the inner part of the tile and linear interpolation for the side part. But in type of tiles, a third case appear. In the corner pixels, no surrounding tiles exists and no interpolation is needed.

### 4.2.2 Pixel Classification Block

The pixel classification block is where the model presented in the previous chapter is implemented. Its objective is to read the statistical information of the image to generate a set of masks which discriminates the pixels with important information and the noise.

![Pixel classification block diagram](Image)

Figure 24: Pixel classification block diagram.

To simplify the work, this block is divided in several sub-blocks, each one with simple tasks to make it easy to update, debug and understand. These blocks are separated depending on their functionality:

- **Histogram generation**: as in the equalization process, the histogram of the image is needed, but in this case is not the histogram of each tile, but of the whole image.

- **Gradient computation**: in order to get an accurate pixel classification, the gradient of the image is needed and computed in this sub-block.

- **Valley detector**: non-low contrast can have histograms in which a division
of the problem is needed. The valley detected is the dividing point of the histogram to solve the problem

- Histogram threshold computation: the model equations are applied in this model in order to compute the correct threshold depending on the statistical information of the image.

- Histogram thresholding: after the threshold is computed, the threshold is applied to the histogram to classify the pixels values into the flat and edge group.

- Pixel grouping: knowing the pixel values that corresponds to each group, a set of sub-masks are generated with the corresponding gradient value that is used in the next block to generate the real mask.

- Classification: having the gradient of the flat and edge groups, the gradient threshold is computed and applied to generate the final set of masks.

After finishing all the blocks, the mask values are stored in 3 memory RAM with 1-bit binary values 0 and 1 in the respective pixel position. The filtering process is applied to the pixel when the value in the mask memory is set to 1 and the pixel is skipped if the value is set to 0.

### 4.2.2.1 Histogram Generation

The histogram of the image is generated with the same components used by the equalization block. This is the `histogram_wrapper` block described in the files `histogram_wrapper_int2.vhd` and `histogram_int3.vhd`.

### 4.2.2.2 Gradient Computation

The gradient is generated by the `derivative_wrapper` block, described in the file `derivative_wrapper.vhd`. It also contains 2 components `sobelkernel` (`sobel_kernel.vhd`) and `serialparallel` (`serial2parallel.vhd`).

To compute the gradient of an image, a kernel is used. In this case, a Sobel 3x3 kernel is used. The block generates 3x3 size windows for each one of the pixels and scans the image in incremental order. When the current central pixel is in the borders of the image, the values corresponding to the window pixels outside the boundaries are set to 0 and the gradient is computed normally with these new values.

The values of the window are introduced one by one to the `serialparallel` component, which keeps them until the window scan is done. Then, all the values are sent
at the same time to the \textit{sobel kernel} component, which receives them and computes the value of the gradient.

\begin{align*}
G_h &= p_3 + 2p_6 + p_9 - p_1 - 2p_4 - p_7 \\
G_v &= p_7 + 2p_8 + p_9 - p_1 - 2p_2 - p_3 \\
G_{d1} &= p_2 + 2p_3 + p_6 - p_4 - 2p_7 - p_8 \\
G_{d2} &= p_4 + 2p_1 + p_2 - p_8 - 2p_9 - p_6 \\
G &= \left( |G_h| + |G_v| + |G_{d1}| + |G_{d2}| \right) / 2
\end{align*}

(19)

In the equation (19), \( p_i \) refers to the pixel position inside the window, being incremented from left to right and top to bottom. The final gradient value has been modified from the standard way of computing it from \( \sqrt{\sum G_i^2} \) to \( \frac{\sum |G_i|}{n/2} \) as a VHDL simplification.

In the gradient threshold computation, the important information is the lower gradient values. In order to save resources, as high values can be automatically considered edges, the gradient is stored in a RAM memory with 8-bit length. All the values resulting from the equation (19) higher than 255 are saturated into that same value.

4.2.2.3 Valley Detector

The valleys are detected with the \textit{valley detection wrapper} block, described in the file \textit{valley_detection_wrapper.vhd}. It also contains 2 sorters (\textit{sorter.vhd}) and a \textit{valley detector} (\textit{valley_detector.vhd}).
The sorters are capable of sorting the first $nbins$ values in the specified order and outputs the entered data in the sorted order and the original position in which entered the component. This way, the same sorter design can be used in different occasions, just changing the parameters.

In order to sort the bins, the algorithm used is the bubble sorter algorithm [10]. This algorithm steps through the list of numbers, comparing each pair and swapping them in the correct order (Figure 26). The process is repeated until all the values in the list are in the correct order.

The first sorter orders all the bins of the histogram in descending order. The first 10 bins are taken and their positions are introduced in the second sorter, which orders the positions in ascending order. After finishing with both sorters, the position of the 10 highest bins are introduced in the valley detector.

Having the positions, the histogram values are read in order from the histogram RAM and introduced into the valley detector. These values are considered local peaks (even if sometimes they are not really peaks when there are 2 consecutive bins). Once the high value bins are loaded, the detector scans in between these bins and checks if there is a histogram bin below 10% of the absolute maximum value.

It is common not to have many valleys, so in order to simplify the design, only 2 valleys are detected.

Figure 26: One iteration of the bubble sort algorithm used in the median computation. The process is repeated until the values are ordered.
4.2.2.4 Histogram Threshold Computation

The parameters necessary to compute the histogram threshold and the threshold itself are computed in the `threshold_histogram_computation` block, described in the file `threshold_histogram_computation.vhd`. It contains 3 components: `threshold_histogram_computation` (threshold_histogram_computation.vhd), `params_computation` (params_computation.vhd) and lastly `histogram_wrapper` (histogram_wrapper_int2.vhd).

This block divides the histogram by the valley point computed in the previous block and introduces the correspondent bin values in the `params_computation` sub-blocks, which its task is to generate the parameters needed ($H_{\text{mean}}$, $H_{\text{median}}$, $H_{\text{max}}$, $H_{\text{min}}$). To obtain the 2 other parameters needed ($H_{\text{grad mean}}$, $H_{\text{grad max}}$), the gradient histogram is obtained with the `histogram_wrapper` and `params_computation` block is used again with the gradient to obtain the parameters.

The maximum and minimum values are easy to obtain. These are stored in the block and every time a value is input, it is compared. If the new value is greater than the current maximum, the new value is stored, and the same process is done with the minimum. Because its simplicity, the output of these 2 parameters are always updated with the correct values.

As for the mean and median, the values are computed once the bin values have been read by the block `params_computation` block. Each time a new value is read, this is summed with all the previous values and a counter is incremented to know how many values have been read. When the block finishes reading all the values, the mean is computed \[ H_{\text{mean}} = \frac{1}{n_{\text{bins}}} \sum x_i \] \[ (20) \]

Once all the parameters are obtained, the `threshold_histogram_computation` starts working to compute the 3 different thresholds according to the model presented in the previous chapter. In this process, fixed point signals are used to be precise in the operations. As VHDL do not implement square root functions, a package has been created (`customized_operations_pkg.vhd`), in which `ufixed_sqrt` function can be found following a non-restoring square root algorithm \[ \text{ufixed_sqrt} \].
4.2.2.5 Histogram Thresholding

The computed thresholds are applied in the histogram_thresholding_wrapper block, described in the file histogram_thresholding_wrapper.vhd. It contains a component histogram_thresholding (histogram_thresholding.vhd).

![Histogram threshold process](image)

Figure 27: Histogram threshold process. Bins above the threshold are considered edge bins and their value in the vector is set to 1. Flat bins below the thresholds are set to 0 in the vector.

The block divides again the histogram in its valley points and introduce the values in a set of histogram_thresholding blocks which are the ones in charge of applying the thresholds. If the bin is above the threshold value, the result is 1, if not it is set to 0. The results for each bin are stored in 256 bits vector, where each position of the vector can be 1 or 0, depending on the result of the threshold operation for that specific bin.

4.2.2.6 Pixel Grouping

The computed thresholds are applied in the pixel_grouping_wrapper block, described in the file pixel_grouping_wrapper.vhd. It contains a component used for each of the thresholds pixel_grouping (pixel_grouping.vhd), which contains all the functionality.

The pixel_grouping process is simple. The gray level value for each one of the pixels is compared with the value in the 256 bits vector, in which the positions indicates if a gray level value is edge or flat. If the position in the vector for the specific pixel is 1 the pixel gradient value is added to the edge group, while the gradient is added to the flat group if the position in the vector for the specific pixel is 0.
Figure 28: Pixel grouping process. The gradient value of edge bin pixels are stored in the edge gradient mask and the rest of pixels are set to 0. Same is done with flat bin pixels in the flat gradient mask.

4.2.2.7 Classification

The final masks are generated by the classification block, described in the file classification.vhd. It contains few component threshold_computation_xxxx_wrapper (threshold_computation_edge_wrapper.vhd), in which xxxx stands for flat or edge, and comparison_binarization (comparison_binarization.vhd).

Figure 29: Classification Block. The masks are generated separately for the edge and flat pixels and then are merged into one final mask.

The threshold computation sub-block is divided in similar 2 different components
because the operations needed to compute the edge and flat gradient thresholds are different. Inside this block, the gradient histogram is obtained and the different thresholds are computed: \( T_h_L \) for the edge group \([21]\) and \( T_h_H \) for the flat one \([22]\), where the positions match with the equations. To do so, the gradient histogram is scan bin by bin until the equation matches. The position in which there is a match is the gradient threshold.

\[
\frac{d^2 H(G_r)}{dG_r^2} = 0 \tag{21}
\]

\[
\left| \frac{dH(G_r)}{dG_r} \right| \approx 0 \tag{22}
\]

Once the thresholds are computed, the gradient pixels are read from the gradient mask memories and compared. The output is set to 1 if the gradient is below the threshold and is set to 0 otherwise. This is done for the edge and flat groups independently and the results of each block are united with an AND gate.

Figure 30: Classification process: the comparison and binarization is done separately and then the results are merged with an AND gate.

From the \textit{pixel\_grouping\_wrapper} block, a pixel not belonging to the group was stored with gradient value 0. Having a gradient 0 is always below the threshold, which results in a mask value of 1 for that group. When the AND gate is applied, the value does not affect the final result and only the value of the other group counts. Later, the filter reads the mask values and it filters the pixel when the mask is set to 1 for the specific pixel.
4.2.3 Discriminative Filter Block

The filter blocks apply a gaussian filter to the image discriminating the pixels protected by the mask. It follows the same architecture as the gradient computation, but in this case the kernel used is not the sobel filter but a gaussian one, and it does not apply the filter operation to all the pixels but only the ones with a mask value set to 1.

The filter entity filtering_wrapper (filtering_wrapper.vhd) includes 2 components that are used several times, one for each mask: serialparallel (serial2parallel.vhd) and filterkernel (filterkernel.vhd).

![Block diagram of the filtering process](image)

Figure 31: Block diagram of the filtering process. The same filter is used several time using different masks as an input.

The block scans the equalized image and the mask positions at the same time in incremental order following the window design structure. The window used is 3x3 size and it is read from left to right and top to bottom. When a window is read, it is shifted to the following pixel, the same way as in the gradient computation. For each iteration, the pixels from a single window are introduced to the serialparallel component. When all the pixels are read, are sent at once to the filterkernel block where the pixel is transformed (23) and then stored to the final memory RAM.

\[
y = \frac{p_1 + 2p_2 + p_3 + 2p_4 + 4p_5 + 2p_6 + p_7 + 2p_8 + p_9}{16}
\]  

(23)

The resulting filtered image is stored in the final memory RAM. Once the first filtering sweep is finished, the process is repeated 2 more times, but instead of obtaining the input values from the equalized image RAM, the values are obtained
from the final memory RAM, where the output of the previous filtering sweep have been stored. Meanwhile, the mask memory is switched to a more restrictive mask, in order to apply the filtering process to the image repeatedly with different protected pixels.

4.3 Implementation results

In order to be able to compare the results, the images used for the FPGA simulation are the same as in the previous chapter, in the Matlab simulations. The image have been transformed with a Matlab script A.2 to a binary file containing all the information of the picture in words of 8 bits length (0-255 range). Different system inputs have been set for each of the images, as each of them have different sizes. Once the simulation has finished, the resulting memory has been read, and the bits from the memory has been transformed into an image with a Matlab script A.3.

The first example image has a width of 382 pixels and a height of 512 pixels. This last one, the limit established by the VHDL programming.

Figure 32: Left is the original image. Right is the image after applying CLAHE.

In the Figure 32 it can be observed how the CLAHE process generates undesired noise in the background. The wall and the coaches have a small amount of gaussian noise and some changes in the tonality which are not gradual. The filtering process has been applied to the image and the results can be observed in Figure 33. The
algorithm makes a good protection, specially in the frontal part where the table and
the flower are. It is not that good in the background part, where the gradient is
severely degraded.

The second example image has a width of 395 pixels and a height of 398 pixels. In
the Figure 34 it can be observed the appearance of few objects not observable in
the first image. The vertebrae are more visible after the equalization and the ribs
are better seen too. It can also be observed how some noise appear in the ribs and
background area. Again, the masks of the discriminative filter do not have as much
precision as the seen in the Matlab simulation. The spinal and ribs area are not as
conservative as in the mathematical simulation and is slightly over-filtered.

In both test images there has been a small degradation of the quality of the
masks. Comparing the implementation in Matlab and a FPGA, there is an impor-
tant different in precision and resolution. While Matlab implementation is based on
the best results possible without taking care about resources, when implementing
the same algorithm in an FPGA some sacrifices should be made, as the hardware
specifications are not unlimited.

In this specific case, most of the limitation is due to the precision of the gradient,
previously described in this chapter. The simplified derivative operation has resulted
in a more lineal and compressed gradient, which has made it more difficult in the

Figure 33: Left is the Matlab simulation of the enhanced image. Right is the FPGA
simulation of the enhanced image.
Figure 34: Left is the original image. Right is the image after applying CLAHE.

Figure 35: Left is the Matlab simulation of the enhanced image. Right is the FPGA simulation of the enhanced image.

pixel classification block to group the pixels but has helped to store the gradient in 8 bit length words of information.
5 Conclusions

5.1 Concluding remarks

The objective of the work presented in this thesis is to improve the a pixel classification algorithm, making it more automated and more conservative, protecting the image information, in order to remove the noise generated in a histogram equalization process, using cascaded low-pass filters. A set of masks are obtained, which are used to protect or expose different areas of the image depending on the level of homogeneity. This masks mark the precision of protection for each of the filters and the quality of the noise removal and signal preservation at the end of the process.

Signal statistics and gradient in the image is used to measure the level of homogeneity of the different areas of the picture. As the images can have a degraded gradient, a coarse classification is made to create a first grouping using the gray level values statistics. This process has been completely automated, analysing the information of the image, specifically of its histogram and computing a set of thresholds without the need of any predefined value.

As the first classification can be slightly rough, a finer grouping is done, using the gradient of the image. Although the gradient can be degraded, because of the first classification, the contribution to the final classification result can be insignificant.

The first coarse classification, in which the work has been focused, groups the pixels based on their homogeneity level taking into consideration that flat areas have big amounts of pixels of the same values. This way, having a histogram of the picture, we can detect where are the pixel values with big or none homogeneity levels. The extreme levels are easy to differentiate, but a threshold should be computed for the intermediate levels. This is done based on a set of parameters obtained from the image histogram, keeping in mind the conservation of image signal.

After analysing the result at the end of section 3 the computed thresholds have been successful in their task to improve the pixel coarse classification. Edges in the resulting image are better protected, even if the characteristics of it are different, while the noise is being removed from the flat areas.

As the Matlab simulations from the proposed algorithm have shown improvement, a FPGA implementation for the whole system has been proposed using VHDL. The initial algorithm has been decomposed in a series of blocks and sub-blocks with specific and simple functionalities, to make the system simple and understandable.

The results of the FPGA implementation have been analysed at the end of section 4. The observed results are not identical to the Matlab simulations due to hardware limitations. When running the first set of simulations, results were the
most important aspect, without taking into account the resources needed, but in real life this is not the way things work. When implementing the algorithm into an FPGA, there are several limitations and parameters we need to fix: memory available, speed, complexity in operations... In this case, the results are optimized depending on the limitations we have.

5.2 Future work

Although the final outcomes are satisfactory, in image processing there is room for more improvement. This can be tackled in both areas developed in this thesis. Looking purely in the mathematically aspect, generating even better result without taking into account the resources, or trying to obtain more optimal results minimizing the hardware resources.

Looking only for the best results, the algorithm presented at the end of section 2 has been improved, creating a model for the histogram threshold using several statistical parameters which already gives good results, but analysing more and new parameters could generate an improvement in the coarse classification. The gradient information is used for the fine classification. As repeatedly said, the gradient is normally severely degraded. Although this degradation has a really small impact on the overall mask generation, an alternative method might be provided avoiding the usage of gradient information, or at least, minimizing it even more.

When trying to minimize resources without degrading the final result some actions can be done. At the end of section 4 problems due to precision and resolution have been presented. This can be fixed increasing the number of bits used to store different parameters and using more complex operations. Doing so, an increase of the memory usage is needed which not always is available and the response time can be increased. There are several parts in which this actions can be done, but not all of them are critical. A study of the components which generate better improvement without sacrificing much resources could be made to get more similar results to the Matlab simulations.
References


A. Matlab Scripts

A.1 Matlab Algorithm Simulation

```matlab
tic;
imname = 'window';
FILTERVAL = [1 1 1 1] / 5;
[LENAall,MAP] = imread(strcat('../../images/','imname','.png'));
LENA = double(LENAall(:, :, 1));
[SZ1,SZ2] = size(LENA);

% Equalization
LENAcla = adapthisteq(uint8(LENA), 'numtiles', [8 8], 'cliplimit', 0.03);
toc

LENA = imfilter(LENA, fspecial('gaussian',[3 3],1));
FX = conv2(LENA, fspecial('sobel'), 'same');
FY = conv2(LENA, fspecial('sobel'), 'same');
FXY1 = conv2(LENA, [0 -1 -2; 1 0 -1; 2 1 0], 'same');
FXY2 = conv2(LENA, fliplr([0 -1 -2; 1 0 -1; 2 1 0]), 'same');
GRAD = sqrt(FX.^2 + FY.^2 + (FXY1.^2 + FXY2.^2) / 2);

FLATMASK1 = zeros(size(LENA));
FLATMASK2 = zeros(size(LENA));
FLATMASK3 = zeros(size(LENA));

LENAHIST = imhist(uint8(LENA));
LENAHIST = conv(LENAHIST, FILTERVAL);
LENAHIST = lenahist/sum(LENAHIST);
LENAHist = lenahist(find(lenahist > 0));
HMAX = max(LENAHist);

% Find valley - has to be less than 10%*Hmax
LENASHORT = sort(lenahist(find(lenahist > 0)), 'descend');
HIGH = lenahist(1:10);
ADD = 0;
for i = 1:10
    POS = find(lenahist == lenahist(i));
    if length(POS) > 1
        PREADD = ADD;
        ADD = ADD + length(POS) - 1;
        HIGHPOS(i+PREADD:i+ADD) = POS;
    else
        HIGHPOS(i+ADD) = POS;
    end
end
HIGHPOS = sort(HIGHPOS, 'ascend');
```
VALLEY(1) = 0;
VALLEYPOS(1) = 1;
for i = 1:length(HIGHPOS)-1
    if HIGHPOS(i+1)-HIGHPOS(i) > 1
        MIN = min(LENAhist(HIGHPOS(i):HIGHPOS(i+1)));
        if MIN < 0.15*HMAX;
            VALLEY(j) = MIN;
            VPOS = find(LENAHIST == MIN);
            VALLEYPOS(j) = VPOS(1);
            j = j+1;
        end
    end
end
VALLEY(j) = 0;
VALLEYPOS(j) = length(LENAHIST);
% Divide the histogram depending on valley
for i = 1:length(VALLEYPOS)-1
    HISTOGRAMS{i} = LENAHIST(VALLEYPOS(i):VALLEYPOS(i+1));
end
% For each subhistogram, compute the threshold
for ihist = 1:length(HISTOGRAMS)
    LENAhist = HISTOGRAMS{ihist}(find(HISTOGRAMS{ihist} > 0));
    MEAN = mean(LENAhist);
    MEDIAN = median(LENAhist);
    HMAX = max(LENAhist);
    HENAHISTORD2 = sort(LENAhist(find(LENAhist > 0), 'ascend'));
    HISTMIN = HENAHISTORD2(floor(length(LENAHISTORD2)*0.2));
    imgradhist = hist(GRAD(:),100);
    imgradhistc = imgradhist(find(max(imgradhist (:)) == imgradhist):
                        length(imgradhist));
    R1grad = mean(imgradhistc (:))/max(imgradhistc (:));
    fixval = R1grad;
    t1 = ((1/(1+2*(MEAN/HMAX)))^1*(1/(1+MEDIAN/MEDIAN))^1+(1+fixval)*(HMAX
        −HISTMIN)/3+HISTMIN;
    t3 = 1/2*((1/(1+(1+(1+MEDIAN/MHMA)^−1)^−2))^1/2)+(1/(1+(1+(1+(1+
        MEDIAN/MEDIAN)^−1)^−2))^1/2)*HMAX;
    t2 = (t1+t3)/2;
    THRESH = [t1 t2 t3];
    FLATGRAY1 = find(HISTOGRAMS{ihist}>(THRESH(1)));
    FLATGRAY2 = find(HISTOGRAMS{ihist}>(THRESH(2)));
    FLATGRAY3 = find(HISTOGRAMS{ihist}>(THRESH(3)));
    for i = 1:ihist-1
        FLATGRAY1 = FLATGRAY1 + length(HISTOGRAMS{i})-1;
        FLATGRAY2 = FLATGRAY2 + length(HISTOGRAMS{i})-1;
        FLATGRAY3 = FLATGRAY3 + length(HISTOGRAMS{i})-1;
    end
    for i = 1:length(FLATGRAY1)
FLATMASK1( uint8 (LENA) == FLATGRAY1(i) + floor (length (FILTERVAL) /2) - 1) = 1;
end
for i = 1:length (FLATGRAY2)
FLATMASK2( uint8 (LENA) == FLATGRAY2(i) + floor (length (FILTERVAL) /2) - 1) = 1;
end
for i = 1:length (FLATGRAY3)
FLATMASK3( uint8 (LENA) == FLATGRAY3(i) + floor (length (FILTERVAL) /2) - 1) = 1;
end
clear VALLEY VALLEYPOS HISTOGRAMS HIGHPOS HIGH ADD
EDGEMASK1 = 1−FLATMASK1;
EDGEMASK2 = 1−FLATMASK2;
EDGEMASK3 = 1−FLATMASK3;
LENAFLAT1 = LENA.*FLATMASK1;
LENAEDGE1 = LENA.*EDGEMASK1;
LENAFLAT2 = LENA.*FLATMASK2;
LENAEDGE2 = LENA.*EDGEMASK2;
LENAFLAT3 = LENA.*FLATMASK3;
LENAEDGE3 = LENA.*EDGEMASK3;
GRADEFAT1 = GRAD.*FLATMASK1;
GRADEDGE1 = GRAD.*EDGEMASK1;
GRADEFAT2 = GRAD.*FLATMASK2;
GRADEDGE2 = GRAD.*EDGEMASK2;
GRADEFAT3 = GRAD.*FLATMASK3;
GRADEDGE3 = GRAD.*EDGEMASK3;
%%% GRAHDIST = GRADEDGE1;
i=0; for thH = (0:0.02:2)*mean(GRAHDIST(GRAHDIST>0))
i = i+1;
TEST = GRAHDIST;
TEST(GRAHDIST<=thH) = 0;
TEST(GRAHDIST>thH+0.02*mean(GRAHDIST(GRAHDIST>0))) = 0;TEST(TEST>0) = 1;
GRAHDIST(i) = sum(TEST(:))/sum(boolean(GRAHDIST(:)));
end

GRADVAR = GRADHDIST(4:i) − GRADHDIST(1:(i−3));
THEEDGE = (find(GRADVAR(3:length(GRADVAR))) == min(GRADVAR(3:length(GRADVAR)))).+1);
THEEDGE1 = (THEEDGE+1)*mean(GRAHDIST(GRAHDIST>0))/50
clear GRADVAR;
GRADM = GRADEDGE2;
i=0; for thM = (0:0.02:2)*mean(GRAHM(GRAHM>0))
i = i + 1;
TEST = GRADM;
TEST(GRADM<=thM) = 0;
TEST(GRADM>thM+0.02*mean(GRADM(GRADM>0))) = 0;TEST(TEST>0)=1;
GRADMDIST(i) = sum(TEST(:))/sum(boolean(GRADM(:))); end

GRADVAR = GRADMDIST(4:i)-GRADMDIST(1:(i-3));
THEDGE = ( find(GRADVAR(3:length(GRADVAR)) == min(GRADVAR(3:length(GRADVAR)))) ,1)+2);
THEDGE2 = (THEDGE+1)*mean(GRADM(GRADM>0))/50
clear GRADVAR;

i=0; for thL = (0:0.02:2)*mean(GRADL(GRADL>0))
i = i + 1;
TEST = GRADL;
TEST(GRADL<=thL) = 0;
TEST(GRADL>thL+0.02*mean(GRADL(GRADL>0))) = 0;TEST(TEST>0)=1;
GRADLDIST(i) = sum(TEST(:))/sum(boolean(GRADL(:))); end

GRADVAR = GRADLDIST(4:i)-GRADLDIST(1:(i-3));
THEDGE = ( find(GRADVAR(3:length(GRADVAR)) == min(GRADVAR(3:length(GRADVAR)))) ,1)+2);
THEDGE3 = (THEDGE+1)*mean(GRADL(GRADL>0))/50
clear GRADVAR GRADHDIST GRADMDIST GRADLDIST ;

EDGEMASK1(GRADEDGE1<THEDGE1) = 0;
GRADFLAT1 = GRAD .* FLATMASK1;
GRADEDGE1 = GRAD .* EDGEMASK1;
EDGEMASK2(GRADEDGE2<THEDGE2) = 0;
GRADFLAT2 = GRAD .* FLATMASK2;
GRADEDGE2 = GRAD .* EDGEMASK2;
EDGEMASK3(GRADEDGE3<THEDGE3) = 0;
GRADFLAT3 = GRAD .* FLATMASK3;
GRADEDGE3 = GRAD .* EDGEMASK3;

GRADH = GRADFLAT1;
i=0; for thH = (0:0.02:2)*mean(GRADH(GRADH>0))
i = i + 1;
TEST = GRADH;
TEST(GRADH<=thH) = 0;
TEST(GRADH>thH+0.02*mean(GRADH(GRADH>0))) = 0;TEST(TEST>0)=1;
GRADHDIST(i) = sum(TEST(:))/sum(boolean(GRADH(:))); end
GRADVAR = GRADHDIST(4:i)−GRADHDIST(1:(i−3));
THFLAT = find(abs(fliplr(GRADVAR))>0.001,1);
if(isempty(THFLAT))
    THFLAT = 0;
end
THFLAT1 = (length(GRADVAR)−THFLAT+3)∗mean(GRADH(GRADH>0))/50
clear GRADVAR;

GRADM = GRADFLAT2;
i=0;for thM = (0:0.02:2)∗mean(GRADM(GRADM>0))
    i = i+1;
    TEST = GRADM;
    TEST(GRADM<=thM) = 0;
    TEST(GRADM>thM+0.02∗mean(GRADM(GRADM>0))) = 0;TEST(TEST>0)=1;
    GRADMDIST(i) = sum(TEST(:))/sum( boolean (GRADM(:)));
end
THFLAT = find(abs(fliplr(GRADVAR))>0.001,1);
if(isempty(THFLAT))
    THFLAT = 0;
end
THFLAT2 = (length(GRADVAR)−THFLAT+3)∗mean(GRADM(GRADM>0))/50
clear GRADVAR;

GRADL = GRADFLAT3;
i=0;for thL = (0:0.02:2)∗mean(GRADL(GRADL>0))
    i = i+1;
    TEST = GRADL;
    TEST(GRADL<=thL) = 0;
    TEST(GRADL>thL+0.02∗mean(GRADL(GRADL>0))) = 0;TEST(TEST>0)=1;
    GRADLDIST(i) = sum(TEST(:))/sum( boolean (GRADL(:)));
end
THFLAT = find(abs(fliplr(GRADVAR))>0.001,1);
if(isempty(THFLAT))
    THFLAT = 0;
end
THFLAT3 = (length(GRADVAR)−THFLAT+3)∗mean(GRADL(GRADL>0))/50
clear GRADVAR GRADHDIST GRADMDIST GRADLDIST ;

FLATMASK1(GRADFLAT1<THFLAT1) = 0;
EDGEMASK1 = EDGEMASK1+FLATMASK1;
GRADFLAT1 = GRAD .∗ FLATMASK1;
A.2 Image to Memory Matlab script

```matlab
% Script to convert bitmaps to .mif format

homefolder = '.../..../images/';
imname = 'xrays_vhdl.png';
image_name = strcat(homefolder,imname);
input_image = imread(image_name);
imgdata=rgb2gray(input_image);
siz = size(imgdata);
```
A.3 Memory to Image Matlab Script

```matlab
% Script to read and show image from RAM dump (.mem Modelsim file)

A = fopen('././memories/out.mem');
B = fgets(A); 
B = fgets(A);
B = fgets(A);

% FLOWER
width = 382;
height = 512;

% XRAYS
width = 395;
height = 398;

C = zeros(height, width);
for i = 1:height
    for j = 1:width
        str = fscanf(A, '%s
', 1);
        isnotbinar = fprintf(str, separation_sym);
        while(~isempty(isnotbinar))
            str = fscanf(A, '%s
', 1);
            isnotbinar = fprintf(str, separation_sym);
        end
        C(height-i+1, width-j+1) = bin2dec(str) / 255;
    end
end
figure;
imshow(C);
```
30 \texttt{fclose} (A)
B Hardware Implementation

B.1 Main Block

main.vhd

```vhdl
library ieee;
library floatfixlib;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use floatfixlib.fixed_pkg.all;
use floatfixlib.math_utility_pkg.all;
entity main is
  port (clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         npixels : in unsigned(18 downto 0);
         x_size : in unsigned(9 downto 0);
         y_size : in unsigned(9 downto 0);
         clip_limit : in unsigned(6 downto 0);
         end_flag : out std_logic);
end main;
architecture wrapper of main is
  -- Components declaration
  -- CLAHE computation
  component clahe_wrapper
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          npixels : in unsigned(18 downto 0);
          x_size : in unsigned(9 downto 0);
          y_size : in unsigned(9 downto 0);
          clip_limit : in unsigned(6 downto 0);
          data_in : in std_logic_vector(7 downto 0);
          addr_im : out std_logic_vector(17 downto 0);
          addr_out : out std_logic_vector(17 downto 0);
          data_out : out std_logic_vector(7 downto 0);
          we : out std_logic;
          end_flag : out std_logic);
  end component;
  component pixelclassification_wrapper
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          data_im : in std_logic_vector(7 downto 0);
          npixels : in unsigned(18 downto 0);
          x_size : in unsigned(9 downto 0);
          y_size : in unsigned(9 downto 0);
          addr_low : out std_logic_vector(17 downto 0);
          addr_mid : out std_logic_vector(17 downto 0);
```

62
addr_high : out std_logic_vector(17 downto 0);
dout_low : out std_logic_vector(0 downto 0);
dout_mid : out std_logic_vector(0 downto 0);
dout_high : out std_logic_vector(0 downto 0);
we_low : out std_logic;
we_mid : out std_logic;
we_high : out std_logic;
end_flag : out std_logic;
addr_im : out std_logic_vector(17 downto 0);
)
end component;

component filtering_wrapper
  port(clk : in std_logic;
  rst : in std_logic;
  en : in std_logic;
  data_im : in std_logic_vector(7 downto 0);
  npixels : in unsigned(18 downto 0);
  x_size : in unsigned(9 downto 0);
  y_size : in unsigned(9 downto 0);
  mask_val : in std_logic;
  addr_im : out std_logic_vector(17 downto 0);
  addr_out : out std_logic_vector(17 downto 0);
  data_out : out std_logic_vector(7 downto 0);
  we : out std_logic;
  end_flag : out std_logic);
end component;

component rom_vhd_mif
  port(clk_i : in std_logic;
  addr_i : in std_logic_vector(17 downto 0);
  data_o : out std_logic_vector(7 downto 0))
end component;

component gradient_ram
  port(
  clk_in : in std_logic;
  clk_out : in std_logic;
  we : in std_logic;
  addr_in : in std_logic_vector(17 downto 0);
  addr_out : in std_logic_vector(17 downto 0);
  data_in : in std_logic_vector(7 downto 0);
  data_out : out std_logic_vector(7 downto 0))
end component;

component mask_ram
  port(
  clk_in : in std_logic;
  clk_out : in std_logic;
  we : in std_logic;
  addr_in : in std_logic_vector(17 downto 0);
  addr_out : in std_logic_vector(17 downto 0);
  data_in : in std_logic_vector(0 downto 0);
  data_out : out std_logic_vector(0 downto 0))
end component;
-- Signal declarations

signal data_im, data_clahe : std_logic_vector(7 downto 0);
signal addr_im, addr_clahe : std_logic_vector(17 downto 0);
signal we_clahe : std_logic;
signal data_out, data_in : std_logic_vector(7 downto 0);
signal addr_out, addr_in : std_logic_vector(17 downto 0);
signal we_mask_low, we_mask_mid, we_mask_high : std_logic;
signal addr_mask_low, addr_mask_mid, addr_mask_high :
  std_logic_vector(17 downto 0);
signal dmask_low, dmask_mid, dmask_high :
  std_logic_vector(7 downto 0);
signal dmask_out_low, dmask_out_mid, dmask_out_high :
  std_logic_vector(7 downto 0);
signal data_filter_in, data_filter, data_filter2 :
  std_logic_vector(7 downto 0);
signal data_fin :
  std_logic_vector(7 downto 0);
signal addr_filter, addr_fin :
  std_logic_vector(17 downto 0);
signal en_filter, mask_data, we_fin, en_filter2 :
  std_logic;
signal mask_num : integer range 0 to 4;
signal end_clahe, end_class, end_filter :
  std_logic;

begin
  clahe : clahe_wrapper
    port map(clk => clk,
             rst => rst,
             en => en,
             npixels => npixels,
             x_size => x_size,
             y_size => y_size,
             clip_limit => clip_limit,
             data_in => data_im,
             addr_im => addr_im,
             addr_out => addr_clahe,
             data_out => data_clahe,
             we => we_clahe,
             end_flag => end_clahe);

  pixelclassification : pixelclassificationwrapper
    port map(clk => clk,
             rst => rst,
             en => en,
             data_im => data_out(7 downto 0),
             npixels => npixels,
             x_size => x_size,
             y_size => y_size,
addr_low <= addr_mask_low,
daddr_mid <= addr_mask_mid,
addr_high <= addr_mask_high,
dout_low <= dmask_low,
dout_mid <= dmask_mid,
dout_high <= dmask_high,
we_low <= we_mask_low,
we_mid <= we_mask_mid,
we_high <= we_mask_high,
end_flag <= end_class,
addr_im <= addr_out
)

filtering : filtering_wrapper
  port map(clk => clk,
            rst => rst,
            en => en_filter2,
            data_im => data_filter_in,
            npixels => npixels,
            x_size => x_size,
            y_size => y_size,
            mask_val => mask_data,
            addr_im => addr_filter,
            addr_out => addr_fin,
            data_out => data_fin,
            we => we_fin,
            end_flag => end_filter);

image_rom : rom_vhd_mif
  port map(clk_i => clk,
            addr_i => addr_out,
            data_o => data_out
  );

image_rom2 : rom_vhd_mif
  port map(clk_i => clk,
            addr_i => addr_im,
            data_o => data_im
  );

clahe_result : gradient_ram
  port map(clk_in => clk,
            clk_out => clk,
            we => we_clahe,
            addr_in => addr_clahe,
            addr_out => addr_filter,
            data_in => data_clahe,
            data_out => data_filter
  );

image_result : gradient_ram
  port map(clk_in => clk,
            clk_out => clk,
            we => we_fin,
            addr_in => addr_fin,
            addr_out => addr_filter,
            data_in => data_fin,
            data_out => data_filter2
  );
mask_ram_low : mask_ram
    port map(clk_in => clk,
      clk_out => clk,
      we => we_mask_low,
      addr_in => addr_mask_low,
      addr_out => addr_filter,
      data_in => dmask_low,
      data_out => dmask_out_low);

mask_ram_mid : mask_ram
    port map(clk_in => clk,
      clk_out => clk,
      we => we_mask_mid,
      addr_in => addr_mask_mid,
      addr_out => addr_filter,
      data_in => dmask_mid,
      data_out => dmask_out_mid);

mask_ram_high : mask_ram
    port map(clk_in => clk,
      clk_out => clk,
      we => we_mask_high,
      addr_in => addr_mask_high,
      addr_out => addr_filter,
      data_in => dmask_high,
      data_out => dmask_out_high);

-- Process to select which mask block is selected
mask_selector : process(clk, en_filter, rst)
begin
  if (clk'event and clk = '1') then
    if (rst = '1' or en_filter = '0') then
      mask_num <= 0;
      end_flag <= '0';
      mask_data <= '0';
      en_filter2 <= '0';
      data_filter_in <= (others => '0');
    else
      -- Starts the filtering process again
      if en_filter2 = '0' then
        en_filter2 <= '1';
      elsif end_filter = '1' and mask_num < 2 then
        mask_num <= mask_num + 1;
        en_filter2 <= '0';
      elsif end_filter = '1' then
        end_flag <= '1';
      end if:
      if mask_num = 0 then
        mask_data <= dmask_out_low(0);
        data_filter_in <= data_filter;
      elsif mask_num = 1 then
        mask_data <= dmask_out_mid(0);
        data_filter_in <= data_filter2;
      else
mask_data <= dmask_out_high(0);
data_filter_in <= data_filter2;
end if;
end if;
end if;
end process;

en_filter <= end_clahe and end_class;
end wrapper;
B.2 CLAHE Block

clahe_wrapper.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clahe_wrapper is
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        npixels : in unsigned(18 downto 0);
        x_size : in unsigned(9 downto 0);
        y_size : in unsigned(9 downto 0);
        clip_limit : in unsigned(6 downto 0);
        data_in : in std_logic_vector(7 downto 0);
        addr_im : out std_logic_vector(17 downto 0);
        addr_out : out std_logic_vector(17 downto 0);
        data_out : out std_logic_vector(7 downto 0);
        we : out std_logic;
        end_flag : out std_logic);
end clahe_wrapper;

architecture wrapper of clahe_wrapper is
  — Components declaration
  component image_tiling
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          npixels : in unsigned(18 downto 0);
          xsize : in unsigned(9 downto 0);
          ysize : in unsigned(9 downto 0);
          addr_im : out std_logic_vector(17 downto 0);
          tilex : out unsigned(2 downto 0);
          tiley : out unsigned(2 downto 0);
          npix_tile : out unsigned(18 downto 0);
          xsize_tile : out unsigned(9 downto 0);
          ysize_tile : out unsigned(9 downto 0);
          ntile : out unsigned(5 downto 0);
          ul_number : out unsigned(6 downto 0);
          ur_number : out unsigned(6 downto 0);
          ll_number : out unsigned(6 downto 0);
          lr_number : out unsigned(6 downto 0);
          up_distance : out unsigned(9 downto 0);
          lo_distance : out unsigned(9 downto 0);
          le_distance : out unsigned(9 downto 0);
          ri_distance : out unsigned(9 downto 0);
          tilechange : out std_logic;
          end_flag : out std_logic);
  end component;
  component histogram_wrapper
    port (clk : in std_logic;

68
reset : in std_logic;
cntr_value : in std_logic_vector(18 downto 0);
pulse_start_input : in std_logic;
im_douta : in std_logic_vector(7 downto 0);
histo_dina : out std_logic_vector(18 downto 0);
histo_addr_a : out std_logic_vector(7 downto 0);
histo_wea : out std_logic_vector(0 downto 0);
histo_addr_b : out std_logic_vector(7 downto 0);
histo_doutb : in std_logic_vector(18 downto 0);
end_flag : out std_logic);
end component;
component histo_ram2
port (clk_in : in std_logic;
clk_out : in std_logic;
we : in std_logic_vector(0 downto 0);
addr_in : in std_logic_vector(7 downto 0);
addr_out : in std_logic_vector(7 downto 0);
data_in : in std_logic_vector(18 downto 0);
data_out : out std_logic_vector(18 downto 0));
end component;
component clahe_clipping
port (clk : in std_logic;
rst : in std_logic;
en : in std_logic;
data_in : in std_logic_vector(18 downto 0);
clip_limit : in unsigned(6 downto 0);
npixels : in unsigned(18 downto 0);
tile : in unsigned(5 downto 0);
addr_in : out std_logic_vector(7 downto 0);
addr_out : out std_logic_vector(7 downto 0);
data_out : out std_logic_vector(18 downto 0);
we : out std_logic;
cdf_min : out std_logic_vector(18 downto 0);
cdf_max : out std_logic_vector(18 downto 0);
change_tile : out std_logic;
end_flag : out std_logic);
end component;
component clahe_equalization is
port (clk : in std_logic;
rst : in std_logic;
en : in std_logic;
image_in : in std_logic_vector(7 downto 0);
addr_in : in std_logic_vector(17 downto 0);
npixels : in unsigned(18 downto 0);
cdf_in_ul : in std_logic_vector(18 downto 0);
cdf_in_ur : in std_logic_vector(18 downto 0);
cdf_in_ll : in std_logic_vector(18 downto 0);
cdf_in_lr : in std_logic_vector(18 downto 0);
cdf_min_ul : in unsigned(18 downto 0);
cdf_min_ur : in unsigned(18 downto 0);
cdf_min_ll : in unsigned(18 downto 0);
cdf_min_lr : in unsigned(18 downto 0);
npixels_ul : in unsigned(18 downto 0);
npixels_ur : in unsigned(18 downto 0);
npixels_ll : in unsigned(18 downto 0);
npixels_lr : in unsigned(18 downto 0);
up_distance : in unsigned(9 downto 0);
lo_distance : in unsigned(9 downto 0);
le_distance : in unsigned(9 downto 0);
ri_distance : in unsigned(9 downto 0);
ul_work : in std_logic;
ur_work : in std_logic;
ll_work : in std_logic;
lr_work : in std_logic;
data_out : out std_logic_vector(7 downto 0);
addr_out : out std_logic_vector(17 downto 0);
we : out std_logic;
end_flag : out std_logic);

end component;

−− Signals
−− Tiling
signal tilex, tiley : unsigned(2 downto 0);
signal npix_tile : unsigned(18 downto 0);
signal xszie_tile, ysize_tile : unsigned(9 downto 0);
signal ntile : unsigned(5 downto 0);
signal tilechange : std_logic;
signal ulnumber, urnumber, llnumber, lrnumber : unsigned(6 downto 0);
signal updistance, lodistance, ledistance, ridistance : unsigned(9 downto 0);
signal addra_hist, addrb_hist : std_logic_vector(7 downto 0);
signal dina_hist, doubt_histo : std_logic_vector(18 downto 0);
signal wea_hist : std_logic_vector(0 downto 0);

−− Histogram computation
type HISTO.IO is array (0 to 63) of std_logic_vector(18 downto 0);
type HISTO.ADDR is array (0 to 63) of std_logic_vector(7 downto 0);
type HISTO.WEA.T is array (0 to 63) of std_logic_vector(0 downto 0);
type TILESIZE is array (0 to 63) of unsigned(18 downto 0);
signal histo_wea : HISTO.WEA.T;
signal histo_addra, histo_addrb : HISTO.ADDR;
signal histo_dina, histo_doubt : HISTO.IO;
signal histo_npix, histo_cdf_min, histo_cdf_max : TILESIZE;
signal addra_histo, addrb_histo : std_logic_vector(7 downto 0);
signal dina_histo, doubt_histo : std_logic_vector(18 downto 0);
signal wea_histo : std_logic_vector(0 downto 0);

−− Clipping
signal addra_clip, addrb_clip : std_logic_vector(7 downto 0);
signal dina_clip, doubt_clip, cdf_min, cdf_max : std_logic_vector(18 downto 0);
signal wea_clip : std_logic_vector(0 downto 0);
signal clip_count : unsigned(5 downto 0);
signal npix_clip : unsigned(18 downto 0);
signal changetile_clip : std_logic;

-- Equalization
signal cdf_in_ul, cdf_in_ur, cdf_in_ll, cdf_in_lr : std_logic_vector(18 downto 0);
signal cdf_min_ul, cdf_min_ur, cdf_min_ll, cdf_min_lr : unsigned(18 downto 0);
signal npixels_ul, npixels_ur, npixels_ll, npixels_lr : unsigned(18 downto 0);
signal ulnumberprev, urnumberprev, llnumberprev, llnumberprev : unsigned(6 downto 0);
signal ulnumberprev2, urnumberprev2, llnumberprev2, llnumberprev2 : unsigned(6 downto 0);
signal updistanceprev, lodistanceprev, ledistanceprev, ridistanceprev : unsigned(9 downto 0);
signal updistanceprev2, lodistanceprev2, ledistanceprev2, ridistanceprev2 : unsigned(9 downto 0);
signal updistanceprev3, lodistanceprev3, ledistanceprev3, ridistanceprev3 : unsigned(9 downto 0);
signal ulwork, urwork, llwork, lrwork : std_logic;
signal addr_equal : std_logic_vector(17 downto 0);
signal data_outt : std_logic_vector(7 downto 0);
--signal addr_out : std_logic_vector(17 downto 0);
signal we_clahe : std_logic;

-- Control signals
signal start_hist : std_logic;
signal en_tiling, en_hist, en_preclip, en_clip, en_equal : std_logic;
signal end_tiling, end_hist, end_clip, end_clip2, end_preclip2, end_equal : std_logic;
signal work_hist : std_logic;

-- ROM ports for test
signal addr_imm : std_logic_vector(17 downto 0);

begin
im_tiling : image_tiling
port map(clk => clk,
          rst => rst,
          ...
en  \Rightarrow en\textunderscore tiling,
npixels \Rightarrow npixels,
xsize  \Rightarrow x\textunderscore size,
ysize  \Rightarrow y\textunderscore size,
addr\textunderscore im \Rightarrow addr\textunderscore imm,
tilex  \Rightarrow tilex,
tiley  \Rightarrow tiley,
npix\textunderscore tile \Rightarrow npix\textunderscore tile,
xsize\textunderscore tile \Rightarrow xsize\textunderscore tile,
ysize\textunderscore tile \Rightarrow ysize\textunderscore tile,
ntile  \Rightarrow ntile,
ul\textunderscore number \Rightarrow ul\textunderscore number,
ur\textunderscore number \Rightarrow ur\textunderscore number,
ll\textunderscore number \Rightarrow ll\textunderscore number,
lr\textunderscore number \Rightarrow lr\textunderscore number,
up\textunderscore distance \Rightarrow up\textunderscore distance,
lo\textunderscore distance \Rightarrow lo\textunderscore distance,
le\textunderscore distance \Rightarrow le\textunderscore distance,
ri\textunderscore distance \Rightarrow ri\textunderscore distance,
tile\textunderscore change \Rightarrow tile\textunderscore change,
end\textunderscore flag  \Rightarrow end\textunderscore tiling
);

histo\textunderscore computation : histogram\_wrapper
  port map( clk  \Rightarrow clk,
reset \Rightarrow rst,
cntv\textunderscore value \Rightarrow std\_logic\_vector(npix\textunderscore tile),
pulse\textunderscore start\_input \Rightarrow start\_histo,
im\textunderscore douna \Rightarrow data\_in,
histo\textunderscore dina \Rightarrow dina\_histo,
histo\textunderscore addra \Rightarrow addra\_histo,
histo\textunderscore wea \Rightarrow wea\_histo,
histo\textunderscore addrb \Rightarrow addrb\_histo,
histo\textunderscore doutb \Rightarrow doutb\_histo,
end\textunderscore flag  \Rightarrow end\_histo
);

tiles : for I in 0 to 63 generate
histo\textunderscore ram : histo\_ram2
  port map( clk\_in  \Rightarrow clk,
clk\_out \Rightarrow clk,
we \Rightarrow histo\_wea(I),
addr\_in \Rightarrow histo\_addra(I),
addr\_out \Rightarrow histo\_addrb(I),
data\_in \Rightarrow histo\_dina(I),
data\_out \Rightarrow histo\_doutb(I)
);
end generate;
clipper : clah\_clipping
  port map( clk  \Rightarrow clk,
rst \Rightarrow rst,
en  \Rightarrow en\_clip,
data\_in \Rightarrow doutb\_clip,
clip\_limit \Rightarrow clip\_limit,
npixels  \Rightarrow npix\_clip,
ntile => clip_count,
addr_in => addrb_clip,
addr_out => addra_clip,
data_out => dina_clip,
we => wea_clip(0),
cdf_min => cdf_min,
cdf_max => cdf_max,
changentile => changentile_clip,
end_flag => end_clip);
equalization : clahe_equalization
  port map(clk => clk,
            rst => rst,
            en => en_equal,
            image_in => data_in,
            addr_in => addr_imm,
npixels => npixels,
cdf_in_ul => cdf_in_ul,
cdf_in_ur => cdf_in_ur,
cdf_in_l1 => cdf_in_l1,
cdf_in_lr => cdf_in_lr,
cdf_min_ul => cdf_min_ul,
cdf_min_ur => cdf_min_ur,
cdf_min_l1 => cdf_min_l1,
cdf_min_lr => cdf_min_lr,
npixels_ul => npixels_ul,
npixels_ur => npixels_ur,
npixels_ll => npixels_ll,
npixels_lr => npixels_lr,
up_distance => updistanceprev3,
lo_distance => lodistanceprev3,
le_distance => ledistanceprev3,
ri_distance => ridistanceprev3,
ul_work => ulwork,
ur_work => urwork,
ll_work => llwork,
lr_work => lrwork,
data_out => data_out,
addr_out => addr_out,
we => we,
end_flag => end_equal);

-- Processes
-- Process to control the connections between the histogram
  computation and
-- the RAM for the specific tile
-- The reading mapping must be updated when the clock goes to 0
-- The writing mapping must be updated when the clock goes to 1
histo_tile : process(clk, rst, en, work_histo)
begin
  if(clk'event and clk = '0') then
if ($rst = '1' \text{ or } \en = '0' \text{ or } (\work_{\histo} = '0' \text{ and } \en_{\equal} = '1'))$ then
  \histo_{\text{dina}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{addr}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{addrb}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{wea}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{npix}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{cdf}_\text{min}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \histo_{\text{cdf}_\text{max}} \leftarrow (\text{others} \Rightarrow (\text{others} \Rightarrow '0'));
  \npix_{\text{clip}} \leftarrow (\text{others} \Rightarrow '0');
  \doutb_{\text{clip}} \leftarrow (\text{others} \Rightarrow '0');

  \text{-- Equalization part}
  \cdf_{\text{min}_{\text{ul}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{min}_{\text{ur}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{min}_{\text{ll}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{min}_{\text{lr}}} \leftarrow (\text{others} \Rightarrow '0');
  \npix_{\text{ul}} \leftarrow (\text{others} \Rightarrow '0');
  \npix_{\text{ur}} \leftarrow (\text{others} \Rightarrow '0');
  \npix_{\text{ll}} \leftarrow (\text{others} \Rightarrow '0');
  \npix_{\text{lr}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{in}_{\text{ul}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{in}_{\text{ur}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{in}_{\text{ll}}} \leftarrow (\text{others} \Rightarrow '0');
  \cdf_{\text{in}_{\text{lr}}} \leftarrow (\text{others} \Rightarrow '0');
  \ulnumber_{\text{prev}} \leftarrow (\text{others} \Rightarrow '1');
  \urnumber_{\text{prev}} \leftarrow (\text{others} \Rightarrow '1');
  \llnumber_{\text{prev}} \leftarrow (\text{others} \Rightarrow '1');
  \lrnumber_{\text{prev}} \leftarrow (\text{others} \Rightarrow '1');
  \ulnumber_{\text{prev2}} \leftarrow (\text{others} \Rightarrow '1');
  \urnumber_{\text{prev2}} \leftarrow (\text{others} \Rightarrow '1');
  \llnumber_{\text{prev2}} \leftarrow (\text{others} \Rightarrow '1');
  \lrnumber_{\text{prev2}} \leftarrow (\text{others} \Rightarrow '1');

  \text{else}
  \text{for } i \text{ in } 0 \text{ to } 63 \text{ loop}
    \text{if } i \text{ to } \text{integer}(\text{ntile}) \text{ and } \en_{\histo} = '0' \text{ then}
      \histo_{\text{dina}}(i) \leftarrow \text{dina}_{\histo};
      \doutb_{\histo} \leftarrow \histo_{\text{doutb}}(i);
      \histo_{\text{addr}}(i) \leftarrow \text{addr}_{\histo};
      \histo_{\text{wea}}(i) \leftarrow \text{wea}_{\histo};
      \histo_{\text{npix}}(i) \leftarrow \npix_{\tile};
    \text{elsif } (i \text{ to } \text{integer}(\text{clip\_count}) \text{ and } \en_{\text{clip}} = '1' \text{ and } \en_{\clip} = '1') \text{ then}
      \histo_{\text{dina}}(i) \leftarrow \text{dina}_{\clip};
      \doutb_{\clip} \leftarrow \histo_{\text{doutb}}(i);
      \histo_{\text{addr}}(i) \leftarrow \text{addr}_{\clip};
      \histo_{\text{wea}}(i) \leftarrow \text{wea}_{\clip};
      \npix_{\clip} \leftarrow \histo_{\npix}(i);
      \histo_{\text{cdf}_\text{min}}(i) \leftarrow \text{unsigned}(\cdf_{\text{min}});
      \histo_{\text{cdf}_\text{max}}(i) \leftarrow \text{unsigned}(\cdf_{\text{max}});
    \text{elsif } (\ulnumber_{\text{prev2}} < '1111111' \text{ and } i \text{ to } \text{integer}(\ulnumber_{\text{prev2}}) \text{ and } \en_{\text{equal}} = '1') \text{ then}
      \cdf_{\text{in}_{\text{ul}}} \leftarrow \histo_{\text{doutb}}(i);
      \cdf_{\text{min}_{\text{ul}}} \leftarrow \histo_{\text{cdf}_\text{min}}(i);
npixels_ul <= histo_cdf_max(i);
elsif(urnumberprev2 < "1111111" and i = to_integer(urnumberprev2) and en_equal = '1') then
cdf_in_ur <= histo_doutb(i);
cdf_min_ur <= histo_cdf_min(i);
npixels_ur <= histo_cdf_max(i);
elsif(llnumberprev2 < "1111111" and i = to_integer(llnumberprev2) and en_equal = '1') then
cdf_in_ll <= histo_doutb(i);
cdf_min_ll <= histo_cdf_min(i);
npixels_ll <= histo_cdf_max(i);
elsif(lrnumberprev2 < "1111111" and i = to_integer(lrnumberprev2) and en_equal = '1') then
cdf_in_lr <= histo_doutb(i);
cdf_min_lr <= histo_cdf_min(i);
npixels_lr <= histo_cdf_max(i);
else
  histo_dina(i) <= (others => '0');
  histo_addra(i) <= (others => '0');
  histo_wea(i) <= (others => '0');
end if;
end loop;
for i in 0 to 63 loop
if i = to_integer(ntile) and end_histo = '0' then
  histo_addrb(i) <= addrb_histo;
elsif(i = to_integer(clip_count) and en_clip = '1' and end_clip2 = '0') then
  histo_addrb(i) <= addrb_clip;
elsif(ulnumber < "1111111" and i = to_integer(ulnumber) and en_equal = '1') then
  histo_addrb(i) <= data_in;
elsif(urnumber < "1111111" and i = to_integer(urnumber) and en_equal = '1') then
  histo_addrb(i) <= data_in;
elsif(llnumber < "1111111" and i = to_integer(llnumber) and en_equal = '1') then
  histo_addrb(i) <= data_in;
elsif(lrnumber < "1111111" and i = to_integer(lrnumber) and en_equal = '1') then
  histo_addrb(i) <= data_in;
else
  histo_addrb(i) <= (others => '0');
end if;
end loop;
if en_equal = '1' then
ulnumberprev <= ulnumber;
urnumberprev <= urnumber;
llnumberprev <= llnumber;
lrnumberprev <= lrnumber;
ulnumberprev2 <= ulnumberprev;
urnumberprev2 <= urnumberprev;
llnumberprev2 <= llnumberprev;
lrnumberprev2 <= lrnumberprev;
else
    ulnumberprev <= (others => '1');
    urnumberprev <= (others => '1');
    llnumberprev <= (others => '1');
    lrnumberprev <= (others => '1');
    ulnumberprev2 <= (others => '1');
    urnumberprev2 <= (others => '1');
    llnumberprev2 <= (others => '1');
    lrnumberprev2 <= (others => '1');
end if;
end if;
end if;
end process;

-- Clip process: Clipping will start for the histogram tiles below
-- ntile
-- (which means that the histogram has been computed). If the
-- clipping
-- process finishes before the histogram computing, it will wait. The
-- mapping
-- with the ram tiles is in histo_tile process
clip_proc : process(clk, rst, en)
begin
  if(clk'event and clk = '1') then
    if(rst = '1' or en = '0') then
      clip_count <= (others => '0');
      en_preclip <= '0';
      en_clip <= '0';
      end_preclip2 <= '0';
      end_clip2 <= '0';
    elsif clip_count < ntile or (clip_count = "111111" and end_tiling = '1') then
      en_clip <= '1';
      if change_tile_clip = '1' and clip_count < "111111" then
        clip_count <= clip_count + 1;
      end if;
      if end_clip = '1' and end_tiling = '1' and end_preclip2 = '0'
      then
        end_preclip2 <= '1';
        elsif end_preclip2 = '1' then
          end_clip2 <= '1';
        end if;
        elsif end_clip2 = '0' then
          en_clip <= '0';
        end if;
    end if;
  end if;
equal_proc : process(clk, rst, en)
begin
  if(clk'event and clk = '0') then
    if(rst = '1' or en = '0' or en_equal = '0') then
      ulwork <= '0';
    end if;
  end if;
end process;
urwork <= '0';
llwork <= '0';
lrwork <= '0';
updistanceprev <= updistance;
lodistanceprev <= lodistance;
ledistanceprev <= ledistance;
ridistanceprev <= ridistance;
updistanceprev2 <= updistanceprev;
lodistanceprev2 <= lodistanceprev;
ledistanceprev2 <= ledistanceprev;
ridistanceprev2 <= ridistanceprev;
updistanceprev3 <= updistanceprev2;
lodistanceprev3 <= lodistanceprev2;
ledistanceprev3 <= ledistanceprev2;
ridistanceprev3 <= ridistanceprev2;
else
  if ulnumberprev2 < "111111" then
    ulwork <= '1';
  else
    ulwork <= '0';
  end if;
if urnumberprev2 < "111111" then
  urwork <= '1';
else
  urwork <= '0';
end if;
if llnumberprev2 < "111111" then
  llwork <= '1';
else
  llwork <= '0';
end if;
if lrnumberprev2 < "111111" then
  lrwork <= '1';
else
  lrwork <= '0';
end if;
updistanceprev <= updistance;
lodistanceprev <= lodistance;
ledistanceprev <= ledistance;
ridistanceprev <= ridistance;
updistanceprev2 <= updistanceprev;
lodistanceprev2 <= lodistanceprev;
ledistanceprev2 <= ledistanceprev;
ridistanceprev2 <= ridistanceprev;
updistanceprev3 <= updistanceprev2;
lodistanceprev3 <= lodistanceprev2;
ledistanceprev3 <= ledistanceprev2;
ridistanceprev3 <= ridistanceprev2;
end if;
end if;
end process;
en_tiling <= (en and (not end_preclip2)) or en_equal;
en_histo <= en;
en_equal  <= en and end_clip2;
work_histo <= en and (not end_tiling) and (not end_histo);
start_histo <= tilechange and (not end_histo);
end_flag  <= end_tiling and en_equal;
addr_im   <= addr_imm;

end architecture wrapper;
B.3 Pixel Classification Block

pixelclassification_wrapper.vhd

library ieee;
library floatfixlib;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use floatfixlib.fixed.pkg.all;
use floatfixlib.math_utility_pkg.all;

entity pixelclassification_wrapper is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_im : in std_logic_vector(7 downto 0);
         npixels : in unsigned(18 downto 0);
         x_size : in unsigned(9 downto 0);
         y_size : in unsigned(9 downto 0);
         addr_low : out std_logic_vector(17 downto 0);
         addr_mid : out std_logic_vector(17 downto 0);
         addr_high : out std_logic_vector(17 downto 0);
         dout_low : out std_logic_vector(0 downto 0);
         dout_mid : out std_logic_vector(0 downto 0);
         dout_high : out std_logic_vector(0 downto 0);
         we_low : out std_logic;
         we_mid : out std_logic;
         we_high : out std_logic;
         end_flag : out std_logic;
         addr_im : out std_logic_vector(17 downto 0)
     ) ;
end pixelclassification_wrapper;

architecture wrapper of pixelclassification_wrapper is

--- Components declaration -----

--- Derivative components ---
component derivative_wrapper
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_im : in std_logic_vector(7 downto 0);
         npixels : in unsigned(18 downto 0);
         x_size : in unsigned(9 downto 0);
         y_size : in unsigned(9 downto 0);
         addr_grad : out std_logic_vector(17 downto 0);
         data_grad : out std_logic_vector(7 downto 0);
         we : out std_logic;
         end_flag : out std_logic;
         addr_im : out std_logic_vector(17 downto 0)
     ) ;
end component;

component gradient_ram
port ( 
  clk_in : in std_logic;  
  clk_out : in std_logic;  
  we      : in std_logic;  
  addr_in : in std_logic_vector(17 downto 0);  
  addr_out: in std_logic_vector(17 downto 0);  
  data_in : in std_logic_vector(7 downto 0);  
  data_out: out std_logic_vector(7 downto 0) 
);  
end component;  

−−−−  Image Histogram Components  −−−−  
component histogram_wrapper  
  port (clk                  : in std_logic;  
        reset                : in std_logic;  
        cntr_value          : in std_logic_vector(18 downto 0);  
        pulse_start_input   : in std_logic;  
        im_douta            : in std_logic_vector(7 downto 0);  
        histo_dina          : out std_logic_vector(18 downto 0);  
        histo_addr           : out std_logic_vector(7 downto 0);  
        histo_wea            : out std_logic_vector(0 downto 0);  
        histo_addrb          : out std_logic_vector(7 downto 0);  
        histo_doutb          : in std_logic_vector(18 downto 0);  
        end_flag            : out std_logic);  
end component;  
component histo_ram2  
  port (clk_in     : in std_logic;  
         clk_out     : in std_logic;  
         we          : in std_logic_vector(0 downto 0);  
         addr_in     : in std_logic_vector(7 downto 0);  
         addr_out    : in std_logic_vector(7 downto 0);  
         data_in     : in std_logic_vector(18 downto 0);  
         data_out    : out std_logic_vector(18 downto 0));  
end component;  
component hist_lowpass_filter is  
  port (clk     : in std_logic;  
         rst      : in std_logic;  
         en       : in std_logic;  
         data_in  : in std_logic_vector(18 downto 0);  
         addr_in  : out std_logic_vector(7 downto 0);  
         addr_out : out std_logic_vector(7 downto 0);  
         data_out : out std_logic_vector(18 downto 0);  
         we       : out std_logic_vector(0 downto 0);  
         end_flag : out std_logic);  
end component;  

−−−−  Valley detection and threshold computation  −−−−  
component valley_detection_wrapper  
  port (clk     : in std_logic;  
         rst      : in std_logic;  
         en       : in std_logic;  
         hist_data_in : in std_logic_vector(18 downto 0);  
         hist_addr  : out std_logic_vector(7 downto 0);
component threshold_histogram_computation_wrapper is
  port ( clk : in std_logic;
                    rst : in std_logic;
                    en : in std_logic;
                    nvalleys : in unsigned(1 downto 0);
                    valley1 : in std_logic_vector(7 downto 0);
                    valley2 : in std_logic_vector(7 downto 0);
                    hist_data_in : in std_logic_vector(18 downto 0);
                    grad_data_in : in std_logic_vector(7 downto 0);
                    nPixels : in unsigned(18 downto 0);
                    TH1 : in std_logic_vector(18 downto 0);
                    TH2 : in std_logic_vector(18 downto 0);
                    TH3 : in std_logic_vector(18 downto 0);
                    TM1 : in std_logic_vector(18 downto 0);
                    TM2 : in std_logic_vector(18 downto 0);
                    TM3 : in std_logic_vector(18 downto 0);
                    TL1 : in std_logic_vector(18 downto 0);
                    TL2 : in std_logic_vector(18 downto 0);
                    TL3 : in std_logic_vector(18 downto 0);
                    hist_addr_in : in std_logic_vector(17 downto 0);
                    hist_addr_in : out std_logic_vector(7 downto 0);
                    end_flag : out std_logic);
end component;
--- component histogram_thresholding
--- port (clk : in std_logic;
--- rst : in std_logic;
--- en : in std_logic;
--- th : in unsigned(18 downto 0); -- High threshold
--- tm : in unsigned(18 downto 0); -- Medium threshold
--- tl : in unsigned(18 downto 0); -- Low threshold
--- hist_data_in : in std_logic_vector(18 downto 0);
--- -- Vectors with edge positions = 1 and flat = 0
--- edge_bin_high : out std_logic_vector(255 downto 0);
--- edge_bin_mid : out std_logic_vector(255 downto 0);
--- edge_bin_low : out std_logic_vector(255 downto 0);
--- hist_addr_in : out std_logic_vector(7 downto 0);
--- end_flag : out std_logic);
--- end component;

component pixel_grouping_wrapper
--- port (clk : in std_logic;
--- rst : in std_logic;
--- en : in std_logic;
--- data_im : in std_logic_vector(7 downto 0);
--- edge_bin_low : in std_logic_vector(255 downto 0);
--- edge_bin_mid : in std_logic_vector(255 downto 0);
--- edge_bin_high : in std_logic_vector(255 downto 0);
--- grad_data_in : in std_logic_vector(7 downto 0);
--- npixels : in unsigned(18 downto 0);
--- grad_edge_low : out std_logic_vector(7 downto 0);
--- grad_flat_low : out std_logic_vector(7 downto 0);
--- grad_edge_mid : out std_logic_vector(7 downto 0);
--- grad_flat_mid : out std_logic_vector(7 downto 0);
--- grad_edge_high : out std_logic_vector(7 downto 0);
--- grad_flat_high : out std_logic_vector(7 downto 0);
--- npixels_edge_low : out unsigned(18 downto 0);
--- npixels_flat_low : out unsigned(18 downto 0);
--- npixels_edge_mid : out unsigned(18 downto 0);
--- npixels_flat_mid : out unsigned(18 downto 0);
--- npixels_edge_high : out unsigned(18 downto 0);
--- npixels_flat_high : out unsigned(18 downto 0);
--- addr_im : out std_logic_vector(17 downto 0);
--- grad_addr_in : out std_logic_vector(17 downto 0);
--- addr_mask : out std_logic_vector(17 downto 0);
--- we : out std_logic;
--- end_flag : out std_logic);
end component;

--- Classification 2 ---
component classification
--- port (clk : in std_logic;
--- rst : in std_logic;
--- en : in std_logic;
--- data_in_edge : in std_logic_vector(7 downto 0);
--- data_in_flat : in std_logic_vector(7 downto 0);
209  npixels  :  in  unsigned(18  downto 0);  
210  npixels_edge :  in  unsigned(18  downto 0);  
211  npixels_flat :  in  unsigned(18  downto 0);  
212  addr_in   :  out  std_logic_vector(17  downto 0);  
213  data_out  :  out  std_logic_vector(0  downto 0);  
214  addr_out  :  out  std_logic_vector(17  downto 0);  
215  we       :  out  std_logic;  
216  end_flag :  out  std_logic);  
217  end  component;  
218  
219  220  Signal  declarations  
221  signal  addr_grad,  addr_grad_out :  std_logic_vector(17  downto 0);  
222  signal  data_grad,  data_grad_out :  std_logic_vector(7  downto 0);  
223  signal  we_grad,  end_derivative :  std_logic;  
224  signal  histo_dina,  histo_dina_histo,  histo_dina_filt,  histo_doutb  
225    :  std_logic_vector(18  downto 0);  
226  signal  histo_addra,  histo_addra_histo,  histo_addra_filt,  histo_addrb,  
227    histo_addrb_histo,  histo_addrb_filt,  histo_addrb_thresh,  
228    histo_addrb_valley :  std_logic_vector(7  downto 0);  
229  signal  start_histo,  end_histo,  histo_started  
230    :  std_logic;  
231  signal  histo_wea,  histo_wea_histo,  histo_wea_filt  
232    :  std_logic_vector(0  downto 0);  
233  signal  end_filt  
234    :  std_logic;  
235  signal  nvalleys  :  unsigned(1  downto 0);  
236  signal  valley1,  valley2 :  std_logic_vector(7  downto 0);  
237  signal  end_valley  :  std_logic;  
238  signal  thhist_addr_grad  
239   (17  downto 0);  
240  signal  thhist_addr_hist  
241   (7  downto 0);  
242  signal  TH1,  TH2,  TH3,  TM1,  TM2,  TM3,  TL1,  TL2,  TL3 :  std_logic_vector  
243    (18  downto 0);  
244  signal  end_thhist_comp  
245    :  std_logic;  
246  signal  end_histo_thresh,  end_pixel_group,  we_group  
247    :  std_logic;  
248  signal  addr_im_group,  addr_grad_group,  addr_group_mask  
249    (17  downto 0);  
250  signal  edge_bin_high,  edge_bin_mid,  edge_bin_low  
251    :  std_logic_vector  
252    (255  downto 0);  
253  signal  grad_edge_low,  grad_flat_low,  grad_edge_mid,  grad_flat_mid,  
254    grad_edge_high,  grad_flat_high  
255    :  std_logic_vector  
256
signal th, tm, tl : unsigned(18 downto 0);  -- Provisional to have some manual thresholds
signal edge_low, edge_mid, edge_high, flat_low, flat_mid, flat_high : std_logic_vector(7 downto 0);
signal npixels_edge_low, npixels_flat_low, npixels_edge_mid, npixels_flat_mid, npixels_edge_high, npixels_flat_high : unsigned(18 downto 0);
signal enable_grad, enable_histo, enable_filt, enable_histo_thresh, enable_pixel_group, enable_class, enable_valley, enable_thhist_comp : std_logic;
signal work_histo, work_filt, work_grad, work_histo_thresh, work_pixel_group, work_class, work_valley, work_thhist_comp : std_logic;
signal addr_im_grad : std_logic_vector(17 downto 0);
signal addr_im_histo : unsigned(17 downto 0);
signal addr_class_low, addr_class_mid, addr_class_high : std_logic_vector(17 downto 0);
signal end_class_low, end_class_mid, end_class_high, end_class : std_logic;

begin

-------- Port mapping ----------------------
-------- Derivative Components --------

derivativewrapper : derivative_wrapper
port map(clk => clk, 
rst => rst, 
en => enable_grad, 
data_im => data_im, 
npixels => npixels, 
x_size => x_size, 
y_size => y_size, 
addr_grad => addr_grad, 
data_grad => data_grad, 
we => we_grad, 
end_flag => end_derivative, 
addr_im => addr_im_grad)
); 
gradientram : gradient_ram

(7 downto 0);
port map( clk_in  => clk,
    clk_out  => clk,
    we     => we_grad,
    addr_in => addr_grad,
    addr_out => addr_grad_out,
    data_in => data_grad,
    data_out => data_grad_out);

--- Image Histogram Components ---

histo : histogram_wrapper
    port map( clk  => clk,
              reset => rst,
              cntr_value => std_logic_vector(npixels),
              pulse_start_input => start_histo,
              im_douta => data_im,
              histo_dina => histo_dina_histo,
              histo_addr.a => histo_addr_histo,
              histo_wea => histo_wea_histo,
              histo_addr.b => histo_addrb_histo,
              histo_doutb => histo_doutb,
              end_flag => end_histo);

histo_ram : histo_ram2
    port map( clk_in  => clk,
              clk_out => clk,
              we     => histo_wea,
              addr_in => histo_addr,
              addr_out => histo_addrb,
              data_in => histo_dina,
              data_out => histo_doutb);

histo_filt : hist_lowpass_filter
    port map( clk  => clk,
              rst => rst,
              en => enable_filt,
              data_in => histo_doutb,
              addr_in => histo_addrb_filt,
              addr_out => histo_addra_filt,
              data_out => histo_dina_filt,
              we => histo_wea_filt,
              end_flag => end_filt);

--- Valley detection and threshold computation ---

valley_detector : valley_detection_wrapper
    port map( clk  => clk,
              rst => rst,
              en => enable_valley,
              hist_data_in => histo_doutb,
              hist_addr => histo_addrb_valley,
              nvalleys => nvalleys,
              valley1 => valley1,
              valley2 => valley2,
end_flag  =>  end_valley
)

thresh_hist_comp : threshold_histogram_computation_wrapper
port map(clk  =>  clk,
     rst  =>  rst,
     en  =>  enable_thhist_comp,
     nvalleys  =>  nvalleys,
     valley1  =>  valley1,
     valley2  =>  valley2,
     hist_data_in  =>  histo_doutb,
     grad_data_in  =>  data_grad_out,
     npixels  =>  npixels,
     TH1  =>  TH1,
     TH2  =>  TH2,
     TH3  =>  TH3,
     TM1  =>  TM1,
     TM2  =>  TM2,
     TM3  =>  TM3,
     TL1  =>  TL1,
     TL2  =>  TL2,
     TL3  =>  TL3,
     grad_addr_in  =>  thhist_addr_grad,
     hist_addr_in  =>  thhist_addr_hist,
     end_flag  =>  end_thhist_comp
);  

--- Histogram thresholding & Pixel grouping Components ---
histo_thresholding : histogram_thresholding_wrapper
port map(clk  =>  clk,
     rst  =>  rst,
     en  =>  enable_histo_thresh,
     nvalleys  =>  nvalleys,
     valley1  =>  valley1,
     valley2  =>  valley2,
     TH1  =>  TH1,
     TH2  =>  TH2,
     TH3  =>  TH3,
     TM1  =>  TM1,
     TM2  =>  TM2,
     TM3  =>  TM3,
     TL1  =>  TL1,
     TL2  =>  TL2,
     TL3  =>  TL3,
     hist_data_in  =>  histo_doutb,
     edge_bin_high  =>  edge_bin_high,
     edge_bin_mid  =>  edge_bin_mid,
     edge_bin_low  =>  edge_bin_low,
     hist_addr_in  =>  histaddrb_thresh,
     end_flag  =>  end_histo_thresh
);  

--- histo_thresholding : histogram_thresholding
--- port map(clk  =>  clk,
---     rst  =>  rst,
---     en  =>  enable_histo_thresh,
th \Rightarrow \text{th},
\text{tm} \Rightarrow \text{tm},
\text{tl} \Rightarrow \text{tl},
\text{hist.data.in} \Rightarrow \text{histo doutb},
\text{edge.bin.high} \Rightarrow \text{edge.bin.high},
\text{edge.bin.mid} \Rightarrow \text{edge.bin.mid},
\text{edge.bin.low} \Rightarrow \text{edge.bin.low},
\text{hist.addr.in} \Rightarrow \text{histo addr thresh},
\text{end.flag} \Rightarrow \text{end_histo_thresh}
)
);

\text{pixel.group} : \text{pixel.grouping.wrapper}

\text{port map}(\text{clk} \Rightarrow \text{clk},
\text{rst} \Rightarrow \text{rst},
\text{en} \Rightarrow \text{enable.pixel.group},
\text{data.im} \Rightarrow \text{data.im},
\text{edge.bin.low} \Rightarrow \text{edge.bin.low},
\text{edge.bin.mid} \Rightarrow \text{edge.bin.mid},
\text{edge.bin.high} \Rightarrow \text{edge.bin.high},
\text{grad.data.in} \Rightarrow \text{data.grad.out},
\text{npixels} \Rightarrow \text{npixels},
\text{grad.edge.low} \Rightarrow \text{grad.edge.low},
\text{grad.flat.low} \Rightarrow \text{grad.flat.low},
\text{grad.edge.mid} \Rightarrow \text{grad.edge.mid},
\text{grad.flat.mid} \Rightarrow \text{grad.flat.mid},
\text{grad.edge.high} \Rightarrow \text{grad.edge.high},
\text{grad.flat.high} \Rightarrow \text{grad.flat.high},
\text{npixels.edge.low} \Rightarrow \text{npixels.edge.low},
\text{npixels.flat.low} \Rightarrow \text{npixels.flat.low},
\text{npixels.edge.mid} \Rightarrow \text{npixels.edge.mid},
\text{npixels.flat.mid} \Rightarrow \text{npixels.flat.mid},
\text{npixels.edge.high} \Rightarrow \text{npixels.edge.high},
\text{npixels.flat.high} \Rightarrow \text{npixels.flat.high},
\text{addr.im} \Rightarrow \text{addr.im.group},
\text{grad.addr.in} \Rightarrow \text{addr.grad.group},
\text{addr.mask} \Rightarrow \text{addr.group.mask},
\text{we} \Rightarrow \text{we.group},
\text{end.flag} \Rightarrow \text{end.pixel.group}
);

\text{mem.grad.edge.low} : \text{gradient.ram}

\text{port map}(\text{clk.in} \Rightarrow \text{clk},
\text{clk.out} \Rightarrow \text{clk},
\text{we} \Rightarrow \text{we.group},
\text{addr.in} \Rightarrow \text{addr.group.mask},
\text{addr.out} \Rightarrow \text{addr.class.low},
\text{data.in} \Rightarrow \text{grad.edge.low},
\text{data.out} \Rightarrow \text{edge.low}
);

\text{mem.grad.edge.mid} : \text{gradient.ram}

\text{port map}(\text{clk.in} \Rightarrow \text{clk},
\text{clk.out} \Rightarrow \text{clk},
\text{we} \Rightarrow \text{we.group},
\text{addr.in} \Rightarrow \text{addr.group.mask},
\text{addr.out} \Rightarrow \text{addr.class.mid},
data_in => grad_edge_mid,
data_out => edge_mid,
);
mem_grad_edge_high : gradient_ram
port map(clk_in => clk,
clk_out => clk,
we => we_group,
addr_in => addr_group_mask,
addr_out => addr_class_high,
data_in => grad_edge_high,
data_out => edge_high);
mem_grad_flat_low : gradient_ram
port map(clk_in => clk,
clk_out => clk,
we => we_group,
addr_in => addr_group_mask,
addr_out => addr_class_low,
data_in => grad_flat_low,
data_out => flat_low);
mem_grad_flat_mid : gradient_ram
port map(clk_in => clk,
clk_out => clk,
we => we_group,
addr_in => addr_group_mask,
addr_out => addr_class_mid,
data_in => grad_flat_mid,
data_out => flat_mid);
mem_grad_flat_high : gradient_ram
port map(clk_in => clk,
clk_out => clk,
we => we_group,
addr_in => addr_group_mask,
addr_out => addr_class_high,
data_in => grad_flat_high,
data_out => flat_high);

--- Classification 2 ---
class_low : classification
port map(clk => clk,
rst => rst,
en => enable_class,
data_in_edge => edge_low,
data_in_flat => flat_low,
npixels => npixels,
npixels_edge => npixels_edge_low,
npixels_flat => npixels_flat_low,
addr_in => addr_class_low,
data_out => dout_low,
addr_out => addr_low,
we \Rightarrow we\_low, 
end\_flag \Rightarrow end\_class\_low 
);
class\_mid : classification 
port map(clk \Rightarrow clk, 
rst \Rightarrow rst, 
en \Rightarrow enable\_class, 
data\_in\_edge \Rightarrow edge\_mid, 
data\_in\_flat \Rightarrow flat\_mid, 
npixels \Rightarrow npixels, 
npixels\_edge \Rightarrow npixels\_edge\_mid, 
npixels\_flat \Rightarrow npixels\_flat\_mid, 
addr\_in \Rightarrow addr\_class\_mid, 
data\_out \Rightarrow dout\_mid, 
addr\_out \Rightarrow addr\_mid, 
we \Rightarrow we\_mid, 
end\_flag \Rightarrow end\_class\_mid 
);
class\_high : classification 
port map(clk \Rightarrow clk, 
rst \Rightarrow rst, 
en \Rightarrow enable\_class, 
data\_in\_edge \Rightarrow edge\_high, 
data\_in\_flat \Rightarrow flat\_high, 
npixels \Rightarrow npixels, 
npixels\_edge \Rightarrow npixels\_edge\_high, 
npixels\_flat \Rightarrow npixels\_flat\_high, 
addr\_in \Rightarrow addr\_class\_high, 
data\_out \Rightarrow dout\_high, 
addr\_out \Rightarrow addr\_high, 
we \Rightarrow we\_high, 
end\_flag \Rightarrow end\_class\_high 
);

−−−−−−− Control the order of the blocks for shared resources −−−−−−−
−−−−−−− 1) Image histogram
−−−−−−− 2) Derivative computation
−−−−−−− 3) Pixel grouping
−−−−−−− 4) Classification

process(clk, rst, en)
  variable thresh\_porc\_low : ufixed(-1 downto -8) := to\_ufixed
    (0.2, -1, -8);  --"000110011";  -- 0.2
  variable thresh\_porc\_mid : ufixed(-1 downto -8) := to\_ufixed
    (0.3, -1, -8);  --"001001101";  -- 0.3
  variable thresh\_porc\_high : ufixed(-1 downto -8) := to\_ufixed
    (0.8, -1, -8);  --"011001101";  --0.8
  variable thvar, tmvar, tlvar : ufixed(18 downto -8);
  variable maxim : ufixed(18 downto 0) := to\_ufixed(0, 18, 0);
begin
  if (clk’event and clk = ’1’) then
if (rst = '1' and en = '0') then
    start_histo <= '0';
    histo_started <= '0';
    addr_im_histo <= (others => '0');
    -- Trial: set maximum value of histogram to 0
    maximhist <= (others => '0');
    th <= "000000010001100110";
    tm <= "000000000000010000";
    -- tl <= "000000000000001000"; — upc logo
    tl <= "000000000010000000100000";
elsif (en'event and en = '1') then
    start_histo <= '1';
    histo_started <= '1';
elsif start_histo = '1' then
    start_histo <= '0';
end if;
if histo_started = '1' then
    addr_im_histo <= addr_im_histo + 1;
    -- Trial: compute the maximum value of the histogram form
    manual threshold
    if maximhist < unsigned(histo_dina) then
        maximhist <= unsigned(histo_dina);
    end if;
    maxim := to_ufixed(maximhist, maxim);
    thvar := maxim * thresh_porc_high;
    tmvar := maxim * thresh_porc_mid;
    tlvar := maxim * thresh_porc_low;
    th <= unsigned(thvar(18 downto 0)); —unsigned(thvar(18 downto 0));
    tm <= unsigned(tmvar(18 downto 0));
    tl <= unsigned(tlvar(18 downto 0));
    —tl <= "00" & maximhist(18 downto 2);
    —tm <= "000" & maximhist(18 downto 3);
end if;
end process;
end_class <= end_class_low and end_class_mid and
end_class_high;
— Enable signals
enable_histo <= en;
enable_grad <= en and end_histo;
enable_filt <= en and end_histo;
enable_valley <= en and end_filt;
enable_thhist_comp <= en and end_valley and end_derivative;
enable_histo_thresh <= en and end_thhist_comp;
enable_pixel_group <= en and end_histo_threshold;
enable_class <= en and end_pixel_group;
— Which block is working
work_histo <= enable_histo and (not end_histo);
work_grad <= enable_grad and (not end_derivative);
work_filt <= enable_filt and (not end_filt);
work_valley <= enable_valley and (not end_valley);
work_thhist_comp <= enable_thhist_comp and (not end_thhist_comp);
work_histo_thresh <= enable_histo_thresh and (not end_histo_thresh);
work_pixel_group <= enable_pixel_group and (not end_pixel_group);
work_class <= enable_class and (not end_class);

addr_im <= std_logic_vector(addr_im_histo) when work_histo = '1' else
    addr_im_grad when work_grad = '1' else
    addr_im_group when work_pixel_group = '1';
addr_grad_out <= addr_grad_group when work_pixel_group = '1' else
    thhist_addr_grad when work_thhist_comp = '1' else
    (others => '0');
histo_addrb <= data_im when work_histo = '1' else
    histo_addrb_filt when work_filt = '1' else
    histo_addrb_valley when work_valley = '1' else
    histo_addrb_thresh when work_histo_thresh = '1' else
    thhist_addr_hist when work_thhist_comp = '1' else
    (others => '0');
histo_addra <= histo_addra_histo when work_histo = '1' else
    histo_addra_filt when work_filt = '1' else
    (others => '0');
histo_dina <= histo_dina_histo when work_histo = '1' else
    histo_dina_filt when work_filt = '1' else
    (others => '0');
histo_wea <= histo_wea_histo when work_histo = '1' else
    histo_wea_filt when work_filt = '1' else
    (others => '0');
end_flag <= end_class;
end wrapper;
B.4 Filtering Block

filtering_wrapper.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filtering_wrapper is
port (clk : in std_logic;
     rst : in std_logic;
     en : in std_logic;
     data_im : in std_logic_vector(7 downto 0);
     npixels : in unsigned(18 downto 0);
     x_size : in unsigned(9 downto 0);
     y_size : in unsigned(9 downto 0);
     mask_val : in std_logic;
     addr_im : out std_logic_vector(17 downto 0);
     addr_out : out std_logic_vector(17 downto 0);
     data_out : out std_logic_vector(7 downto 0);
     we : out std_logic;
     end_flag : out std_logic);
end filtering_wrapper;

architecture wrapper of filtering_wrapper is
-- Components declaration
component serialparallel
port (clk : in std_logic;
     rst : in std_logic;
     en : in std_logic;
     data_in : in std_logic_vector(7 downto 0);
     p1 : out std_logic_vector(7 downto 0);
     p2 : out std_logic_vector(7 downto 0);
     p3 : out std_logic_vector(7 downto 0);
     p4 : out std_logic_vector(7 downto 0);
     p5 : out std_logic_vector(7 downto 0);
     p6 : out std_logic_vector(7 downto 0);
     p7 : out std_logic_vector(7 downto 0);
     p8 : out std_logic_vector(7 downto 0);
     p9 : out std_logic_vector(7 downto 0);
     end_flag : out std_logic);
end component;
component filterkernel is
port (clk : in std_logic;
     rst : in std_logic;
     en : in std_logic;
     start : in std_logic;
     p1 : in std_logic_vector(7 downto 0);
     p2 : in std_logic_vector(7 downto 0);
     p3 : in std_logic_vector(7 downto 0);
     p4 : in std_logic_vector(7 downto 0);
     p5 : in std_logic_vector(7 downto 0);
```

92
p6 : in std_logic_vector(7 downto 0);
p7 : in std_logic_vector(7 downto 0);
p8 : in std_logic_vector(7 downto 0);
p9 : in std_logic_vector(7 downto 0);
mask_val : in std_logic;
data_out : out std_logic_vector(7 downto 0);
we : out std_logic;
end_flag : out std_logic);
end component;

-- Signals declarations
signal pixel1, pixel2, pixel3, pixel4, pixel5, pixel6, pixel7, pixel8,
pixel9 : std_logic_vector(7 downto 0);
signal start_filter :
  : std_logic;
signal finish_filter :
  : std_logic;
signal end_reading, end_block :
  : std_logic;
signal data_in :
  : std_logic_vector(7 downto 0);
signal im_addr :
  : std_logic_vector(19 downto 0);
signal addrout :
  : unsigned(17 downto 0);
signal x_pos, y_pos :
  : unsigned(9 downto 0);  -- indexes in the
image map
signal x_tile, y_tile :
  : integer range 0 to 2;  -- indexes inside the
tile
signal border :
  : std_logic;  -- specifies if the border tile
is inside the
-- image or not
signal maskval :
  : std_logic;

begin
serial_parallel : serialparallel
  port map(clk => clk,
    rst => rst,
    en => en,
    data_in => data_im,
p1 => pixel1,
p2 => pixel2,
p3 => pixel3,
p4 => pixel4,
p5 => pixel5,
p6 => pixel6,
p7 => pixel7,
p8 => pixel8,
p9 => pixel9,
end_flag => start_filter);
filter_kernel : filterkernel
  port map(clk => clk,
    rst => rst,
    en => en,
    start => start_filter,
--- Compute the addresses to get the image pixels.
--- The addr counter will move around all the pixels of the image, and for
--- each position will take it's 3x3 tile neighborhood.
--- In the corners, the values of the tiles outside the image range will be 0

process (clk, rst)
begin
    if (clk'event and clk = '1') then
        if (rst = '1' or en = '0') then -- restart all the procedure
            x_pos <= (others => '0');
            y_pos <= (others => '0');
            x_tile <= 0;
            y_tile <= 0;
            border <= '0';
            im_addr <= (others => '0');
            maskval <= '0';
            end_flag <= '0';
        elsif end_flag = '0' then
            if x_tile < 2 then
                x_tile <= x_tile + 1;
            elsif y_tile < 2 then
                -- Increase y_tile and reset x_tile
                y_tile <= y_tile + 1;
                x_tile <= 0;
            elsif x_pos < (x_size - 1) then
                -- Increase x_pos and reset x_tile, y_tile
                x_pos <= x_pos + 1;
                x_tile <= 0;
                y_tile <= 0;
            elsif y_pos < (y_size - 1) then
                -- Increase y_pos and reset x_tile, y_tile, x_pos
                y_pos <= y_pos + 1;
                x_tile <= 0;
                y_tile <= 0;
                x_pos <= (others => '0');
            else
                -- end
                end_flag <= '1';
        end if;
    end if;
end process;
end if;

-- Border outsiders control
if x_pos = 0 and x_tile = 0 then
    border <= '1';
elsif x_pos = x_size and x_tile = 2 then
    border <= '1';
elsif y_pos = 0 and y_tile = 0 then
    border <= '1';
elsif y_pos = y_size and y_tile = 2 then
    border <= '1';
else
    border <= '0';
end if;

-- Get the mask value
if (x_tile = 1 and y_tile = 1) then
    maskval <= mask_val;
end if;

-- Address computation
im_addr <= std_logic_vector((y_pos-1+y_tile) * x_size + (x_pos-1+x_tile));
end if;
end if;
end process;

-- Compute the addresses to store the gradient pixels. The value will be
-- stored each time the finish_filter and we is set to 1
process(clk, rst, end_reading)
begin
    if (clk’event and clk = '1') then
        if (rst = '1' or en = '0') then  -- restart all the parameters
            addrout <= (others => '0');
            end_block <= '0';
        elsif (finish_filter = '1' and end_reading = '0') then
            addrout <= addrout + 1;
            end_block <= '1';
        elsif (finish_filter = '1' and end_reading = '1') then
            addrout <= addrout + 1;
            end_block <= '1';
        end if;
    end if;
end process;

addr_out <= std_logic_vector(addrout);
end_flag <= end_block;

-- Control the borders: when pixel is a border and the tile is
-- outside the
-- image range, return a zeros as a image value
addr_im <= im_addr(17 downto 0) when (border = '0') else
    (others => '0') when (border = '1');
197  data_in \leq data_im \text{ when (} \text{border} = '0' \text{) else (others} \Rightarrow '0') \text{ when (} \text{border} = '1'\text{);} \\
199  \text{end wrapper;}
B.5 Image Tiling

image_tiling.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- This block divides the image in 8x8 blocks and outputs the address of the pixel in the memory, the size of the tiles, the tile number in which the pixel is situated, the ul, ur, ll, lr tile numbers and the distances from their center points

entity image_tiling is
port(clk : in std_logic;
rst : in std_logic;
en : in std_logic;
npixels : in unsigned(18 downto 0);
xsize : in unsigned(9 downto 0);
ysize : in unsigned(9 downto 0);
addr_im : out std_logic_vector(17 downto 0);
tilex : out unsigned(2 downto 0);  -- Tile number x
tiley : out unsigned(2 downto 0);  -- Tile number y
npx_tile : out unsigned(18 downto 0);  -- #pixels in tile
xsize_tile : out unsigned(9 downto 0);
ysize_tile : out unsigned(9 downto 0);
tile : out unsigned(5 downto 0);
-- Tile number of the ul, ur, ll and lr tiles
-- If (other => '1') ul or ur or ll or lr tile
ul_number : out unsigned(6 downto 0);
ur_number : out unsigned(6 downto 0);
ll_number : out unsigned(6 downto 0);
lr_number : out unsigned(6 downto 0);
-- Distance from the current pixel position to the center of the tiles
up_distance : out unsigned(9 downto 0);
lo_distance : out unsigned(9 downto 0);
le_distance : out unsigned(9 downto 0);
ri_distance : out unsigned(9 downto 0);
tilechange : out std_logic;
d_flag : out std_logic);
end image_tiling;

architecture Behavioral of image_tiling is
-- Signals
signal addrout : unsigned(22 downto 0);
signal addrtilestart : unsigned(22 downto 0);
signal addrx, addry : unsigned(9 downto 0);
signal started, finish : std_logic;
signal tilechg : std_logic;
 Signals related to tiling size and number

\[
\begin{align*}
signal xsize\_tile, ysize\_tile : & \quad \text{unsigned(9 downto 0)}; \\
signal numx, numy : & \quad \text{unsigned(2 downto 0)}; \\
signal npx\_tile : & \quad \text{unsigned(19 downto 0)}; \\
signal numy1 : & \quad \text{unsigned(3 downto 0)}; \\
signal n\_tiles : & \quad \text{unsigned(7 downto 0)};
\end{align*}
\]

 Signals for the nearby tiles and distances

\[
\begin{align*}
signal ul\_number, ur\_number, ll\_number, lr\_number : & \quad \text{unsigned(6 downto 0)}; \\
signal up\_distance, lod\_distance, led\_distance, rid\_distance : & \quad \text{unsigned(9 downto 0)};
\end{align*}
\]

begin

-- addrx increases
-- when addrx above size\_tile -> addry increases
-- when addry above size\_tile -> tile number x increases
-- when numx above number of tiles -> tile number y increases
-- when numy above number of tiles -> process finishes
-- the final address is computed at the bottom

addrproc : process(clk, rst, en)
begin

if (clk'event and clk = '1') then
  if (rst = '1' or en = '0') then
    addrx <= (others => '0');
    addry <= (others => '0');
    numx <= (others => '0');
    numy <= (others => '0');
    tilechg <= '1';
    started <= '0';
    finish <= '0';
  elsif started = '0' then
    if tilechg = '0' then
      tilechg <= '1';
    else
      tilechg <= '0';
      started <= '1';
    end if;
  elsif addrout < npx\_pixels then
    if addrx < (xsize\_tile - 1) then
      addrx <= addrx + 1;
    elsif addry < (ysize\_tile - 1) then
      addry <= addry + 1;
      addrx <= (others => '0');
    elsif numx < 7 then
      numx <= numx + 1;
      addrx <= (others => '0');
      addry <= (others => '0');
      started <= '0';
    elsif numy < 7 then
      numy <= numy + 1;
      numx <= (others => '0');
      addrx <= (others => '0');
  end if;
end process;
addry <= (others => '0');
start <= '0';

else
finish <= '1';
end if;
end if;
end if;
end process;

-- The image will be divided in a grid of 8x8 tiles
-- The size for each tile will be computed, for the first tiles the size will
-- be always the same, for the last tiles xtile = 8 and ytile = 8,
-- might be smaller
compute_tile_size : process(rst, en, xsize, ysize, npixels)
variable stdsizex, stdsizey : unsigned(9 downto 0) := (others => '0');
variable xsizetilevar, ysizetilevar : unsigned(19 downto 0) := (others => '0');
begin
if (rst = '1' or en = '0') then
xsizetile <= (others => '0');
ysizetile <= (others => '0');
npixtile <= (others => '0');
stdsizex := (others => '0');
stdsizey := (others => '0');
else
stdsizex := xsize/8;
stdsizey := ysize/8;
-- Std size for < 8 and smaller sized in needed for = 8
if numx < 7 and numy < 7 then
xsizetile <= stdsizex;
ysizetile <= stdsizey;
npixtile <= stdsizex*stdsizey;
else
xsizetilevar := xsize - stdsizex*8;
ysizetilevar := ysize - stdsizey*8;
xsizetile <= xsizetilevar(9 downto 0);
ysizetile <= ysizetilevar(9 downto 0);
npixtile <= xsizetilevar(9 downto 0) * ysizetilevar(9 downto 0);
end if;
end if;
end if;
end process;

-- Process that computes the nearby tiles number and the distance from it
-- 1) Detect in which quarter of the tile is the pixel situated
-- 2) Depending of the quarter, get the nearby tile numbers
-- 3) Compute the distances
near_tiles : process(clk, rst, en, addrx, addry, xsizetile, ysizetile)
begin
  if (clk 'event and clk = '0') then
    if (rst = '1' or en = '0') then
      ulnumber <= (others => '1');
      urnumber <= (others => '1');
      llnumber <= (others => '1');
      lnumber <= (others => '1');
      updistance <= (others => '0');
      lodistance <= (others => '0');
      ledistance <= (others => '0');
      ridistance <= (others => '0');
    else
      if (addrx < xsize tile/2 and addry < ysize tile/2) then -- UL
        quarter
        if (numx = 0 and numy = 0) then -- Corner
          ulnumber <= (others => '1');
          urnumber <= (others => '1');
          llnumber <= (others => '1');
          lnumber <= ntile (6 downto 0);
        elsif numx = 0 then -- Left Side
          ulnumber <= (others => '1');
          urnumber <= ntile (6 downto 0) - 8;
          llnumber <= (others => '1');
          lnumber <= ntile (6 downto 0);
        elsif numy = 0 then -- Upper Side
          ulnumber <= ntile (6 downto 0) - 9;
          urnumber <= ntile (6 downto 0) - 8;
          llnumber <= ntile (6 downto 0) - 1;
          lnumber <= ntile (6 downto 0);
        else
          ulnumber <= ntile (6 downto 0); urnumber <= ntile (6 downto 0) - 8;
          llnumber <= ntile (6 downto 0) - 1;
          lnumber <= ntile (6 downto 0);
        end if;
        updistance <= ysize tile/2 + addry;
        lodistance <= ysize tile/2 - addry;
        ledistance <= xsize tile/2 + addrx;
        ridistance <= xsize tile/2 - addrx;
      elsif (addrx > xsize tile/2-1 and addry < ysize tile/2) then -- UR
        quarter
        if (numx = 7 and numy = 0) then -- Corner
          ulnumber <= (others => '1');
          urnumber <= (others => '1');
          llnumber <= ntile (6 downto 0);
          lnumber <= (others => '1');
        elsif numx = 7 then -- Right Side
          ulnumber <= ntile (6 downto 0) - 8;
          urnumber <= (others => '1');
          llnumber <= ntile (6 downto 0);
          lnumber <= (others => '1');
        elsif numy = 0 then -- Upper Side
          ulnumber <= (others => '1');
      end if;
    end if;
  end if;
end begin;
urnumber <= (others => '1');
llnumber <= ntile(6 downto 0);
lrnumber <= ntile(6 downto 0) + 1;
else
  ulnumber <= ntile(6 downto 0) - 8;
  urnumber <= ntile(6 downto 0) - 7;
  llnumber <= ntile(6 downto 0);
lrnumber <= ntile(6 downto 0) + 1;
end if;
updistance <= ysize/2 + addy;
lodistance <= ysize/2 - addy;
ledistance <= addrx - xsize/2;
ridistance <= xsize + xsize/2 - addrx;
elif(addrx < xsize/2 and addy > ysize/2-1) then --
  LL quarter
  if(numx = 0 and numy = 7) then   -- Corner
    ulnumber <= (others => '1');
    urnumber <= ntile(6 downto 0);
    llnumber <= (others => '1');
lrnumber <= (others => '1');
  elsif numx = 0 then            -- Left Side
    ulnumber <= (others => '1');
    urnumber <= ntile(6 downto 0);
    llnumber <= (others => '1');
lrnumber <= ntile(6 downto 0) + 8;
  elsif numy = 7 then             -- Bottom Side
    ulnumber <= ntile(6 downto 0) - 1;
    urnumber <= ntile(6 downto 0);
    llnumber <= (others => '1');
lrnumber <= (others => '1');
  else
    ulnumber <= ntile(6 downto 0) - 1;
    urnumber <= ntile(6 downto 0);
    llnumber <= ntile(6 downto 0) + 7;
lrnumber <= ntile(6 downto 0) + 8;
  end if;
  updistance <= addy - ysize/2;
  lodistance <= ysize + ysize/2 - addy;
  ledistance <= xsize/2 + addrx;
  ridistance <= xsize + xsize/2 - addrx;
else -- LR quarter
  if(numx = 7 and numy = 7) then   -- Corner
    ulnumber <= ntile(6 downto 0);
    urnumber <= (others => '1');
    llnumber <= (others => '1');
lrnumber <= (others => '1');
  elsif numx = 7 then            -- Right Side
    ulnumber <= ntile(6 downto 0);
    urnumber <= (others => '1');
    llnumber <= ntile(6 downto 0) + 8;
lrnumber <= (others => '1');
  elsif numy = 7 then             -- Bottom Side
    ulnumber <= ntile(6 downto 0);
urnumber <= ntiles(6 downto 0) + 1;
llnumber <= (others => '1');
lrnumber <= (others => '1');
else
  ulnumber <= ntiles(6 downto 0);
  urnumber <= ntiles(6 downto 0) + 1;
  llnumber <= ntiles(6 downto 0) + 8;
  lrnumber <= ntiles(6 downto 0) + 9;
end if;
updistance <= addry - ysize_tile/2;
loadistance <= ysize_tile + ysize_tile/2 - addry;
ledistance <= addrx - xsize_tile/2;
ridistance <= xsize_tile + xsize_tile/2 - addrx;
end if;
end if;
end if;
end process;

-- Computed address to obtain pixel from ROM
adrtilestart <= (xsize_tile * numx) + (xsize * ysize_tile * numy);
addrout <= adrtilestart + (addry * xsize) + addrx;
numy1 <= '0' & numy;
nTiles <= numy1*8 + numx;

-- Mapping the signals with the ports
addr_im <= std_logic_vector(addrout(17 downto 0));
tilex <= numx;
tiley <= numy;
xsize_tile <= xsize_tile;
ysize_tile <= ysize_tile;
npix_tile <= npix_tile(18 downto 0);
ntile <= ntiles(5 downto 0);
ul_number <= ulnumber;
ur_number <= urnumber;
ll_number <= llnumber;
lr_number <= lrnumber;
up_distance <= updistance;
lo_distance <= ledistance;
le_distance <= ledistance;
ri_distance <= ridistance;
tilechange <= tilechg;
end_flag <= finish;
end architecture Behavioral;
B.6 Histogram Wrapper

histogram_wrapper_int2.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity histogram_wrapper is
  port(
    -- global clock signal, active with its rising edge
    clk : in std_logic;
    -- reset signal, synchronous and active high
    reset : in std_logic;
    -- number of pixels of the image
    cntr_value : in std_logic_vector(18 downto 0);
    -- Trigger to start the histogram generation
    pulse_start_input : in std_logic;
    -- Output of the histogram data
    histogram_out: out std_logic_vector(17 downto 0);
    -- im_addr : out std_logic_vector(17 downto 0);
    -- enable : in std_logic;
    im_douta : in std_logic_vector(7 downto 0);
    histo_dina : out std_logic_vector(18 downto 0);
    histo_addr : out std_logic_vector(7 downto 0);
    histo_wea : out std_logic_vector(0 downto 0);
    histo_addrb : out std_logic_vector(7 downto 0);
    histo doutb : in std_logic_vector(18 downto 0);
    -- histo rstb : out std_logic;
    end_flag : out std_logic
  );
end histogram_wrapper;

architecture wrapper of histogram_wrapper is
  signal device_data : std_logic_vector(7 downto 0);  -- current pixel value
  signal sel_data_input : std_logic;  -- selector between histogram
generation/reading modes
  signal ram_wr_addr : std_logic_vector(7 downto 0);  -- address to be accessed in the RAM containing the histogram
  signal wren : std_logic_vector(0 downto 0);  -- histogram ram write enable
  signal dataout : std_logic_vector(18 downto 0);  -- output of the histogram counters values
  signal pulse_out, pulse_out_2, pulse_out_3 : std_logic;
  signal pulse_out, rstb, end_flag_signal : std_logic;
signal image_addr : std_logic_vector(3 downto 0);
signal image_addr : unsigned(17 downto 0);
signal histo_out : std_logic_vector(18 downto 0);

constant IMAGE_PIXELS : integer := 262143;

component histogram --computes the histogram
  port ( addrin : in std_logic_vector(7 downto 0); -- device
data as address for RAM
datain : in std_logic_vector (18 downto 0); -- RAM data
out
  clk : in std_logic;
  cntr_value : in std_logic_vector (18 downto 0);
  ramwraddr : out std_logic_vector(7 downto 0);
  rstcntr : in std_logic;
  rst : in std_logic;
  start_cntr : in std_logic;
  wren : out std_logic;
  data_out : out std_logic_vector(18 downto 0); -- RAM data
  end_flag : out std_logic
);
end component;

begin
--image_rom, histogram_generator and histogram_ram are interconnected
 according to the design
--principle proposed in:
--http://www.edn.com/design/integrated-circuit-design/4363979/Compute-a
--histogram—in—an—FPGA—with—one—clock
--However, some functionality was simplified or removed because it was
 not needed.

histogram_generator : histogram
port map(
  addrin  => device_data,
datain   => histo_out,
clk      => clk,
cntr_value => cntr_value,
ramwraddr => ram_wr_addr,
rst      => reset,
start_cntr => pulse_out,
wren     => wren(0),
data_out => dataout,
end_flag => end_flag_signal
);


process(clk) -- Process containing an address counter to read the
image in the
-- ROM memory sequentially and compute its histogram
begin
  if (CLK'event and CLK = '1') then
    -- if reset = '1' or pulse_out = '1' then
      -- image_addr <= to_unsigned(0, 18);
    -- else
      -- image_addr <= image_addr + 1;
    -- end if;
    if reset = '1' then
      pulse_out <= '0';
    else
      pulse_out <= pulse_start_input;
    end if;
  end if;
end process;
-- histogram_out <= histo_out;
rstb <= reset or pulse_out;
-- im_addr <= std_logic_vector(image_addr);
device_data <= im_douta;
histo_dina <= dataout;
histo_addr <= ram_wr_addr;
histo_wea <= wren;
-- histo_rst <= rstb;
histo_addrb <= device_data;
histo_out <= histo_doutb;
end_flag <= end_flag_signal;
-- end_flag <= '0';
end wrapper;
B.7 Histogram

histogram_int3.vhd

---2013/04/20---Forked from the description available at:
---http://www.edn.com/design/integrated-circuit-design/4363979/Compute-a
− histogram—-in—-an—-FPGA—-with—-one—-clock

---2013/05/01---The code has been simplified to remove unneeded
functionality and make interfacing easier.

---2013/05/03---Fixed a bug which caused histogram count increasing to
not work properly when
two or more consecutive pixels with the same exact value appeared.
---2013/05/04---Comments added for clarity and future reference.
---2013/05/07---More bugfixes, related to component reset.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity histogram is
port (addrin : in std_logic_vector(7 downto 0); — device data
as address for RAM
datain : in std_logic_vector (18 downto 0); — RAM data
out
cnt_value : in std_logic; —Synchronous rising edge clock
pixels of the input image
cntr : std_logic_vector(18 downto 0); —Number of pixels
ramwraddr : out std_logic_vector(7 downto 0); —Address
where the updated histogram value must be
written. Equal to the grey value
(from 0 to 255)
start_cnt : in std_logic; —global reset
wren : out std_logic; —write enable output for the ram
containing the histogram
data_out : out std_logic_vector(18 downto 0); — RAM data
in
end_flag : out std_logic
);

architecture hlsm of histogram is
signal wr_addr, wr_addr1
: std_logic_vector(7 downto 0);
signal pre_cnt, next_cnt, pre_dout, dout
: std_logic_vector(18 downto 0); — count no. of
samples for which histogram to be computed.
signal pre_addrcnt, next_addrcnt
    : std_logic_vector(7 downto 0);
signal addr, pre_addr
    : std_logic_vector(7 downto 0);
signal end_flag_signal, wren_signal, wren_next, wren_next1,
    wren_next2, addrpreadr : std_logic;

begin

addr <= addrin;

process (clk, rst)
begin
    if (clk'event and clk = '1') then
        if (rst = '1' or start_cnt = '1') then
            -- restart all the procedure
            pre_cntr <= (others => '0');
            wren_next1 <= '0';
            wren_next <= '0';
            wren_signal <= '0';
            wren <= '0';
            pre_addr <= (others => '0');
            addrpreadr <= '0';
            wr_addr1 <= (others => '0');
            wr_addr <= (others => '0');
            end_flag_signal <= '0';
        else
            pre_cntr <= next_cnt;
            wren_next1 <= wren_next2;
            wren_next <= wren_next1;  -- delay write enable changes to sync it
            wren <= wren_next;  -- with the output of valid values
            wren_signal <= wren_next;
            pre_addr <= addrin;  -- store current pixel gray value and its associated
            pre_dout <= dout;  -- counter for use if next pixel’s gray value is equal
        end if;
        if wren_signal = '1' and wren_next = '0' then
            end_flag_signal <= '1';
        else
            end_flag_signal <= end_flag_signal;
        end if;
        if (addr = pre_addr) then
            -- if the gray value is the same of the
            -- previous clock, load pre_dout instead of RAM in
            addrpreadr <= '1';
        end if;
    end if;
end process;

else
    addrpreaddr <= '0';
end if;

end if;

wr_addr1 <= addr;
wr_addr <= wr_addr1;     -- delay write address by 2
    clock
end if;
end process;

process(datain, addrpreaddr, rst, pre_cntr)
begin
    if((pre_cntr >= cntr_value) or active='0') then
        if((pre_cntr >= cntr_value)) then      -- finish if the internal counter
            next_cntr <= pre_cntr;           -- amount of pixels in the photo
            wren_next2 <= '0';
        else
            wren_next2 <= '1';               -- else: keep calculating and
            writing.
            next_cntr <= pre_cntr + '1';
        end if;
    if(rst = '1') then
        dout <= (others => '0');
    else
        if(datain = "1111111111111110") then   -- prevent overflow
            dout <= datain;
        elsif addrpreaddr = '1' then          -- See previous process for
            dout <= pre_dout + 1;             -- addrpreaddr's "if/else" functionality
        end if;
        else
            dout <= datain + '1';
        end if;
    end if;
    ramwraddr <= wr_addr;
    data_out <= dout;
end process;
end_flag <= end_flag_signal;
end hlsm;
B.8 CLAHE Clipping

clahe_clipping.vhd

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;

4 entity clahe_clipping is
5   port (clk : in std_logic;
6          rst : in std_logic;
7          en : in std_logic;
8          data_in : in std_logic_vector(18 downto 0);
9          clip_limit : in unsigned(6 downto 0);
10         npixels : in unsigned(18 downto 0);
11         ntile : in unsigned(5 downto 0);
12         addr_in : out std_logic_vector(7 downto 0);
13         addr_out : out std_logic_vector(7 downto 0);
14         data_out : out std_logic_vector(18 downto 0);
15         we : out std_logic;
16         cdf_min : out std_logic_vector(18 downto 0);
17         cdf_max : out std_logic_vector(18 downto 0);
18         changetile : out std_logic;
19         end_flag : out std_logic)
20 end clahe_clipping;

21 architecture Behavioral of clahe_clipping is
22
23   begin
24
25     — Signals
26     signal addrin, addrout, addrpre1, addrpre2 : unsigned(7 downto 0);
27     signal dout : unsigned(18 downto 0);
28     signal excess : unsigned(18 downto 0);
29     signal abs_clip : unsigned(25 downto 0);
30     signal ex_redistribute : unsigned(18 downto 0);
31     signal end_excess, end_cdf, finish : std_logic;
32     signal wren : std_logic;
33     signal cdfmin, cdfmax : unsigned(18 downto 0);
34
35     begin
36
37     — Scan the addresses
38     — 1st scan: compute the excess with the clip
39     — 2nd scan: redistribute excess and compute cdf
40     addr_comp : process(clk, rst, en)
41     begin
42     if (clk'event and clk = '1') then
43     if (rst = '1' or en = '0') then
44     addrin <= (others => '0');
45     addrout <= (others => '0');
46     end_cdf <= '0';
47     end_excess <= '0';
48     finish <= '0';
49     elsif addrin < "11111111" then
50     addrin <= addrin + 1;
51     if end_excess = '0' then

addrpre1 <= addrin;
addrpre2 <= addrpre1;
addrout  <= addrpre2;
else
    addrout  <= addrin;
end if;
elsif addrout < "11111111" then
    addrpre1 <= addrin;
    addrpre2 <= addrpre1;
    addrout  <= addrpre2;
elsif end_excess = '0' then
    end_excess <= '1';
    addrin  <= (others => '0');
    addrout  <= addrin;
elsif end_excess = '1' and end_cdf = '0' then
    end_cdf <= '1';
    addrout  <= addrin;
else
    end_cdf <= '0';
    end_excess <= '0';
    addrin  <= (others => '0');
    addrout  <= (others => '0');
    if ntile = "111111" then
        finish <= '1';
    end if;
end if;
en end if;
en end process;

— Compute excess and subtract it from the entrance if the datain is above
— the clip limit. If datain > clip then in writes in memory clip value and
— keeps the excess for the 2nd scan.
— In the 2nd scan (end_excess='1') the redistributed excess is added to the
— bins of the histogram
excess_comp : process(clk, rst, en)
variable doutvar : unsigned(37 downto 0);
begin
    if clk'event and clk = '1' then
        if (rst = '1' or en = '0') then
            excess  <= (others => '0');
            dout    <= (others => '0');
            cdfmin  <= (others => '0');
            cdfmax  <= (others => '0');
            doutvar := (others => '0');
            wren    <= '0';
        elsif end_excess = '0' then
            if unsigned(data_in) > abs_clip then
                excess <= excess + (unsigned(data_in) - abs_clip(18 downto 0));
            end if;
            dout    <= abs_clip(18 downto 0);
        end if;
    end if;
end if;
en end process;
wren <= '1';
else
    dout <= unsigned(data_in);
    wren <= '0';
end if;
elsif end_excess = '1' and end_cdf = '0' then
    doutvar := dout + (unsigned(data_in) + ex_redistribute) * 256 / npixels;
    dout <= doutvar(18 downto 0);
    if addrin = "00000000" then
        cdfmin <= doutvar(18 downto 0);
    elsif addrout = "11111111" then
        cdfmax <= doutvar(18 downto 0);
    end if;
    wren <= '1';
else
    end_cdf = '1' then
        wren <= '0';
        excess <= (others => '0');
        dout <= (others => '0');
        cdfmin <= (others => '0');
        doutvar := (others => '0');
    end if;
end if;
end process;

-- Real clip limit
abs_clip <= npixels * clip_limit / 100;
-- Quantity to redistribute to each bin
ex_redistribute <= excess / 256;
data_out <= std_logic_vector(dout);
addr_in <= std_logic_vector(addrin);
addr_out <= std_logic_vector(addrout);
cdf_min <= std_logic_vector(cdfmin);
cdf_max <= std_logic_vector(cdfmax);
we <= wren;
change_tile <= end_cdf;
end_flag <= finish;
end architecture Behavioral;
B.9 CLAHE Transformation

clahe_equalization.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clahe_equalization is
  port(clk : in std_logic;
    rst : in std_logic;
    en : in std_logic;
    image_in : in std_logic_vector(7 downto 0);
    addr_in : in std_logic_vector(17 downto 0);
    npixels : in unsigned(18 downto 0);
    -- Data for each of the tiles
    -- u:upper; l:lower; l;left; r:right
    cdf_in_ul : in std_logic_vector(18 downto 0);
    cdf_in_ur : in std_logic_vector(18 downto 0);
    cdf_in_ll : in std_logic_vector(18 downto 0);
    cdf_in_lr : in std_logic_vector(18 downto 0);
    cdf_min_ul : in unsigned(18 downto 0);
    cdf_min_ur : in unsigned(18 downto 0);
    cdf_min_ll : in unsigned(18 downto 0);
    cdf_min_lr : in unsigned(18 downto 0);
    npixels.ul : in unsigned(18 downto 0);
    npixels.ur : in unsigned(18 downto 0);
    npixels.ll : in unsigned(18 downto 0);
    npixels.lr : in unsigned(18 downto 0);
    up_distance : in unsigned(9 downto 0);
    lo_distance : in unsigned(9 downto 0);
    le_distance : in unsigned(9 downto 0);
    ri_distance : in unsigned(9 downto 0);
    ul_work : in std_logic;
    ur_work : in std_logic;
    ll_work : in std_logic;
    lr_work : in std_logic;
    data_out : out std_logic_vector(7 downto 0);  -- Pixel value after CLAHE
    addr_out : out std_logic_vector(17 downto 0);  --addr for the CLAHE RAM
    we : out std_logic;
    end_flag : out std_logic);
  end clahe_equalization;

architecture Behavioral of clahe_equalization is
  -- Signals
  signal addrpre1, addrpre2, addrpre3, addrpre4, addrout :
    std_logic_vector(17 downto 0);
  signal eq.ul, eq.ur, eq.ll, eq.lr :
    unsigned(37 downto 0);
  signal dataout :
    unsigned(37 downto 0);
signal finish, wren : std_logic;
signal eq_ulur, eq ulll, eq ullr, eq urll, eq urlr, eq lllr :
unsigned(47 downto 0);
signal eq urlllr, eq ullllr, eq ulurlr, eq ulurll :
unsigned(47 downto 0);
signal updistance, lodistance, ledistance, ridistance :
unsigned(9 downto 0);
— retard signals
signal updistance2, lodistance2, ledistance2, ridistance2 :
unsigned(9 downto 0);
signal cdfminul, cdfminur, cdfminll, cdfminlr :
unsigned(18 downto 0);
signal npixelsul, npixelsur, npixelsll, npixelslr :
unsigned(18 downto 0);
—
signal eq_all :
unsigned(57 downto 0);

begin — Process to keep the address to store in the final memory
addrproc : process(clk, rst, en)
begin
  if (clk'event and clk = '1') then
    if (rst = '1' or en = '0') then
      addrpre1 <= (others => '0');
      addrpre2 <= (others => '0');
      addrpre3 <= (others => '0');
      addrpre4 <= (others => '0');
      addrout <= (others => '0');
      wren <= '0';
      finish <= '0';
      updistance <= up_distance;
      lodistance <= lo_distance;
      ledistance <= le_distance;
      ridistance <= ri_distance;
      updistance2 <= updistance;
      lodistance2 <= lodistance;
      ledistance2 <= ledistance;
      ridistance2 <= ridistance;
      cdfminul <= cdf min ul;
      cdfminur <= cdf min ur;
      cdfminll <= cdf min ll;
      cdfminlr <= cdf min lr;
      npixelsul <= npixels ul;
      npixelsur <= npixels ur;
      npixelsll <= npixels ll;
      npixelslr <= npixels lr;
    elsif unsigned(addr_in) < npixels - 1 then
      updistance <= up_distance;
      lodistance <= lo_distance;
      ledistance <= le_distance;
      ridistance <= ri_distance;
  end if;
end process addrproc;
equalproc : process(clk, rst, en)
begin
  if (clk'event and clk = '1') then
    if (rst = '1' or en = '0') then
      eq_ul <= (others => '0');
      eq_ur <= (others => '0');
      eq_ll <= (others => '0');
      eq_lr <= (others => '0');
    else
      if ul_work = '1' then
        eq_ul <= (unsigned(cdf_in_ul) - cdf_min_ul)*255/(npiexls_ul - cdf_min_ul);
      else
        eq_ul <= (others => '0');
      end if;
      if ur_work = '1' then
        eq_ur <= (unsigned(cdf_in_ur) - cdf_min_ur)*255/(npiexls_ur - cdf_min_ur);
      else
        eq_ur <= (others => '0');
      end if;
      if ll_work = '1' then
        eq_ll <= (others => '0');
      else
        eq_ll <= (others => '0');
      end if;
      if rr_work = '1' then
        eq_rr <= (others => '0');
      else
        eq_rr <= (others => '0');
      end if;
    end if;
  end if;
end process;
eq_ll <= (unsigned(cdf_in_ll) - (cdf_min_ll)*255/(npixels_ll - cdf_min_ll));
eq ll <= ( others => '0' );
end if;
if lr_work = '1' then
  eq_lr <= (unsigned(cdf_in_lr) - (cdf_min_lr)*255/(npixels_lr - cdf_min_lr));
else
  eq_lr <= ( others => '0' );
end if;
end if;
end process;

-- All possible combinations for the equalization

eq_ulur <= (eq_ul*(ridistance) + eq_ur*(ledistance))/(ridistance + ledistance);
eq_uull <= (eq_ul*(lodistance) + eq_ll*(updistance))/(lodistance + updistance);
eq_urlr <= (eq_ur*(lodistance) + eq_lr*(updistance))/(lodistance + updistance);
eq_lllr <= (eq_ll*(ridistance) + eq_lr*(ledistance))/(ridistance + ledistance);
eq ullam <= (eq_ll*(lodistance) + eq_lr*(ledistance))/(ridistance + ledistance);
eq_urlr <= (eq_ul*(lodistance+ridistance) + eq_lr*(updistance+ledistance))/(lodistance+ridistance+updistance+ledistance);
eq //////////////////////////////////////////////////////////////////////

115
eq_ulur(37 downto 0) when ul_work = '1' and ur_work = '1' and ll_work = '0' and lr_work = '0' else
  eq_uull(37 downto 0) when ul_work = '1' and ur_work = '0'
  eq_uullr(37 downto 0) when ul_work = '1' and ur_work = '0'
  eq_uurl(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uulll(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uullrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uurlr(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrr(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrr(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrrr(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurrr(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uurlrr(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrl(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrrl(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrrll(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uurlrrrrl(37 downto 0) when ul_work = '1' and ur_work = '1'
  eq_uulrr(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrrrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrrrrl(37 downto 0) when ul_work = '0' and ur_work = '0'
  eq_uulrrrrrrl(37 downto 0) when ul_work = '0' and ur_work = '0'

end architecture Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity serialparallel is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector (7 downto 0);
         p1 : out std_logic_vector (7 downto 0);
         p2 : out std_logic_vector (7 downto 0);
         p3 : out std_logic_vector (7 downto 0);
         p4 : out std_logic_vector (7 downto 0);
         p5 : out std_logic_vector (7 downto 0);
         p6 : out std_logic_vector (7 downto 0);
         p7 : out std_logic_vector (7 downto 0);
         p8 : out std_logic_vector (7 downto 0);
         p9 : out std_logic_vector (7 downto 0);
         end_flag : out std_logic );
end serialparallel;

architecture Behavioral of serialparallel is
  signal counter : integer range 0 to 9;
signal out1, out2, out3, out4, out5, out6, out7, out8, out9 :
  std_logic_vector (7 downto 0);
signal finish :
  std_logic;
begin
process (clk, rst)
begin
  if (clk'event and clk = '1') then
    if (rst = '1' or en = '0') then    --restart all the procedure
      counter <= 0;
      finish <= '0';
      out1 <= (others => '0');
      out2 <= (others => '0');
      out3 <= (others => '0');
      out4 <= (others => '0');
      out5 <= (others => '0');
      out6 <= (others => '0');
      out7 <= (others => '0');
      out8 <= (others => '0');
      out9 <= (others => '0');
    elsif en = '1' then
      if counter < 9 then
        finish <= '0';
        counter <= counter + 1;
        out9 <= data_in;
      end if;
      finish <= '0';
      counter <= counter + 1;
      out9 <= data_in;
    end if;
  end if;
end process;
...
out8 <= out9;
out7 <= out8;
out6 <= out7;
out5 <= out6;
out4 <= out5;
out3 <= out4;
out2 <= out3;
out1 <= out2;
end if;
if counter = 8 then
  finish <= '1';
counter <= 0;
end if;
end if;
end process;
p1 <= out1;
p2 <= out2;
p3 <= out3;
p4 <= out4;
p5 <= out5;
p6 <= out6;
p7 <= out7;
p8 <= out8;
p9 <= out9;
end_flag <= finish;
end Behavioral;
B.11 Sobel Kernel

sobel_kernel.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

-- sobel kernel will compute the sobel gradient of the tile 3x3 in 1 clock
entity sobelkernel is
port (clk : in std_logic;
  rst : in std_logic;
  en : in std_logic;
  start : in std_logic;
  p1 : in std_logic_vector(7 downto 0);
  p2 : in std_logic_vector(7 downto 0);
  p3 : in std_logic_vector(7 downto 0);
  p4 : in std_logic_vector(7 downto 0);
  p5 : in std_logic_vector(7 downto 0);
  p6 : in std_logic_vector(7 downto 0);
  p7 : in std_logic_vector(7 downto 0);
  p8 : in std_logic_vector(7 downto 0);
  p9 : in std_logic_vector(7 downto 0);
  grad_out : out std_logic_vector(7 downto 0);
  we : out std_logic;
  end_flag : out std_logic)
);

architecture Behavioral of sobelkernel is
signal in1, in2, in3, in4, in5, in6, in7, in8, in9 : std_logic_vector(7 downto 0);
signal started, loaded, finish : std_logic;
begin
  process(clk, rst, en, start)
  variable summax, summay, summay1, summay2 : std_logic_vector(10 downto 0);
  variable summa1, summa2, summa3, summa4 : std_logic_vector(10 downto 0);
  variable summa : unsigned(10 downto 0);
  begin
    if (clk'event and clk = '1') then
      if (rst = '1' or en = '0') then
        -- restart all the procedure
        finish <= '0';
        we <= '0';
        started <= '0';
        loaded <= '0';
        grad_out <= (others => '0');
        in1 <= (others => '0');
        in2 <= (others => '0');
        in3 <= (others => '0');
      else
        ... (remaining code continues here)
      end if
    end if
  end process
end sobelkernel;
in4 <= (others => '0');
in5 <= (others => '0');
in6 <= (others => '0');
in7 <= (others => '0');
in8 <= (others => '0');
in9 <= (others => '0');

elsif (en = '1' and start = '1') then
    start <= '1';
elsif (en = '1' and started = '1' and loaded = '0') then
    loaded <= '1';
in1 <= p1;
in2 <= p2;
in3 <= p3;
in4 <= p4;
in5 <= p5;
in6 <= p6;
in7 <= p7;
in8 <= p8;
in9 <= p9;
elsif (en = '1' and loaded = '1') then
    summax := ("000" & in3) + ("00" & in6 & '0') + ("000" & in9) - ("000" & in1) - ("00" & in4 & '0') - ("000" & in7);
    summay := ("000" & in7) + ("00" & in8 & '0') + ("000" & in9) - ("000" & in1) - ("00" & in2 & '0') - ("000" & in3);
    summaxy1 := ("000" & in2) + ("00" & in3 & '0') + ("000" & in6) - ("000" & in4) - ("00" & in7 & '0') - ("000" & in8);
    summaxy2 := ("000" & in4) + ("00" & in1 & '0') + ("000" & in2) - ("000" & in8) - ("00" & in9 & '0') - ("000" & in6);

    -- Absolute values
    if summax(10) = '1' then
        summa1 := not summax+1;
    else
        summa1 := summax;
    end if;
    if summay(10) = '1' then
        summa2 := not summay+1;
    else
        summa2 := summay;
    end if;
    if summaxy1(10) = '1' then
        summa3 := not summaxy1+1;
    else
        summa3 := summaxy1;
    end if;
    if summaxy2(10) = '1' then
        summa4 := not summaxy2+1;
    else
        summa4 := summaxy2;
    end if;

    summa := (unsigned(summa1)+unsigned(summa2)+unsigned(summa3)+unsigned(summa4))/2;
-- Threshold = 255
if summa > "000001111111" then
  grad_out <= (others => '1');
else
  grad_out <= std_logic_vector(summa(7 downto 0));
end if;
we <= '1';
finish <= '1';
started <= '0';
loaded <= '0';
elif finish = '1' then
  finish <= '0';
  we   <= '0';
end if;
end if;
end process;
end_flag <= finish;
end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity valley_detection_wrapper is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         hist_data_in : in std_logic_vector(18 downto 0);
         hist_addr : out std_logic_vector(7 downto 0);
         nvalleys : out unsigned(1 downto 0);
         valley1 : out std_logic_vector(7 downto 0);
         valley2 : out std_logic_vector(7 downto 0);
         end_flag : out std_logic ) ;
end valley_detection_wrapper;

architecture wrapper of valley_detection_wrapper is
  ———— Components declaration ————
component sorter
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(18 downto 0);
         nbins : in unsigned(7 downto 0);
         order : in std_logic;
         data_out : out std_logic_vector(18 downto 0);
         pos_out : out std_logic_vector(7 downto 0);
         end_sort : out std_logic;
         end_flag : out std_logic ) ;
end component;
component valley_detector
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(7 downto 0);
         max : in unsigned(18 downto 0);
         hist_in : in std_logic_vector(18 downto 0);
         hist_addr : out std_logic_vector(7 downto 0);
         nvalleys : out unsigned(1 downto 0);
         valley1 : out std_logic_vector(7 downto 0);
         valley2 : out std_logic_vector(7 downto 0);
         end_flag : out std_logic ) ;
end component;
signal dout_sort1, dout_sort2, din_sort2 :
  std_logic_vector(18 downto 0);
signal pout_sort1, pout_sort2 :
  std_logic_vector(7 downto 0);
signal end_sort1, end_sort2, end_flag_sort1, end_flag_sort2 :
  std_logic;
signal addr_sort :
  unsigned(7 downto 0);
signal addr_detect :
  std_logic_vector(7 downto 0);
signal localmax :
  unsigned(18 downto 0);

begin
  ----------------- Port mapping -----------------
  sort1 : sorter
    port map(clk => clk,
              rst => rst,
              en => en,
              data_in => hist_data_in,
              nbins => to_unsigned(255, 8),
              order => '1',
              data_out => dout_sort1,
              pos_out => pout_sort1,
              end_sort => end_sort1,
              end_flag => end_flag_sort1);
  sort2 : sorter
    port map(clk => clk,
              rst => rst,
              en => end_sort1,
              data_in => din_sort2,
              nbins => to_unsigned(8, 8),
              order => '0',
              data_out => dout_sort2,
              pos_out => pout_sort2,
              end_sort => end_sort2,
              end_flag => end_flag_sort2);
  detector : valley_detector
    port map(clk => clk,
              rst => rst,
              en => end_sort2,
              data_in => dout_sort2(7 downto 0),
              max => localmax,
              hist_in => hist_data_in,
              hist_addr => addr_detect,
              nvalleys => nvalleys,
              valley1 => valley1,
              valley2 => valley2,
              end_flag => end_flag);

  process(clk, rst)
  begin
if clk = '1' and clk'event then
  if rst = '1' or en = '0' then
    addr_sort <= (others => '0');
    localmax <= (others => '0');
    elsif end_sort1 = '0' then
      addr_sort <= addr_sort + 1;
      if localmax < unsigned(hist_data_in) then
        localmax <= unsigned(hist_data_in);
        end if;
    end if;
  end if;
end process;

hist_addr <= std_logic_vector(addr_sort) when end_sort1 = '0' else
            addr_detect;
bin_sort2 <= "00000000000" & pout_sort1;
end wrapper;
B.13 Valley Detector

valley_detector.vhd

```vhdl
library ieee;
library floatfixlib;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use floatfixlib.fixed_pkg.all;
use floatfixlib.math_utility_pkg.all;

entity valley_detector is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(7 downto 0);
         max : in unsigned(18 downto 0);
         hist_in : in std_logic_vector(18 downto 0);
         hist_addr : out std_logic_vector(7 downto 0);
         valleys : out unsigned(1 downto 0);
         valley1 : out std_logic_vector(7 downto 0);
         valley2 : out std_logic_vector(7 downto 0);
         end_flag : out std_logic);
end entity valley_detector;

architecture Behavioral of valley_detector is
  type data_array is array(0 to 10) of std_logic_vector(7 downto 0);
  type valley_array is array(0 to 1) of std_logic_vector(7 downto 0);
  signal data : data_array;
  signal valleys : valley_array;
  signal addr, addr_delta, addr_prev : unsigned(7 downto 0);
  signal localmin : unsigned(18 downto 0);
  signal localminpos : std_logic_vector(7 downto 0);
  signal finish_load, finish : std_logic;
  signal counter, counternext : integer range 0 to 12;
  signal nval : unsigned(1 downto 0);
  signal thresh : ufixed(18 downto -9);
  signal maxthresh : ufixed(18 downto -9);

begin
  process(clk, rst)
  begin
    if clk = '1' and clk'event then
      if rst = '1' or en = '0' then
        data <= (others => (others => '0'));
        addr <= (others => '0');
        addr_prev <= (others => '0');
        addr_delta <= (others => '0');
        localmin <= (others => '1');
        localminpos <= (others => '0');
        valleys <= (others => (others => '1'));
        finish_load <= '0';
      else
```

125
finish <= '0';
counter <= 0;
counternext <= 1;
nval <= (others => '0');
thresh <= to_ufixed(max, 18, 0) * to_ufixed(0.1, -1, -9);
maxthresh <= to_ufixed(max, 18, 0) * to_ufixed(0.7, -1, -9);
elsif counter < 10 and finish load = '0' then
  -- Load the 10 histogram bins to the array
  data(counter) <= data_in;
counter <= counter + 1;
localmin <= (others => '1');
localminpos <= (others => '1');
elsif unsigned(addr) < unsigned(data(9)) and finish <= '0' then
  elsif unsigned(addr) < unsigned(data(7)) and finish <= '0' then
    -- Find the valleys
    if finish load = '0' then
      finish load <= '1';
counter <= 0;
addr <= unsigned(data(0));
elsif (addr = unsigned(data(counter)) and unsigned(hist_in) < unsigned(maxthresh(18 downto 0))) then
  addr_delta <= (others => '0');
counter <= counter + 1;
  counternext <= counter + 2;
  addr <= unsigned(data(counter+1));
  -- Scan the addresses between peak bins
  -- If the minimum value between the 2 bins is less than 0.1*max
  -- then it will be a valley
  elsif addr < unsigned(data(counternext)) then
    addr <= unsigned(data(counter))+addr_delta;
    addr_delta <= addr_delta+1;
    addr_prev <= addr;
    if unsigned(hist_in) < localmin then
      -- Keep the lowest value
      localmin <= unsigned(hist_in);
      localminpos <= std_logic_vector(addr);
      end if;
elsif (addr = unsigned(data(counternext)) and unsigned(hist_in) < unsigned(maxthresh(18 downto 0))) then
  addr_delta <= (others => '0');
counter <= counter + 1;
  counternext <= counternext + 1;
else
  addr_delta <= (others => '0');
counter <= counter + 1;
  counternext <= counter + 2;
  if localmin < unsigned(thresh(18 downto 0)) then
    -- If min is below 0.15*max, then is a valley
    valleys(to_integer(nval)) <= localminpos;
    localmin <= (others => '1');
nval <= nval + 1;
  end if;
end if;
if nval = "10" then
    finish <= '1';
    end if;
else
    finish <= '1';
    end if;
end if;
end process;

nvalleys <= nval;
valley1 <= valleys(0);
valley2 <= valleys(1);
hist_addr <= std_logic_vector(addr);
end_flag <= finish;

end architecture Behavioral;
**B.14 Sorter**

**sorter.vhd**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sorter is
  port (clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(18 downto 0);
         nbins : in unsigned(7 downto 0);
         order : in std_logic;
         data_out : out std_logic_vector(18 downto 0);
         pos_out : out std_logic_vector(7 downto 0);
         end_sort : out std_logic;
         end_flag : out std_logic);
end entity sorter;

architecture Behavioral of sorter is
  signal finish_insert, finish_sort, finish : std_logic;
  signal counter : integer range 0 to 255;
  signal i, j : integer range 0 to 255;
  signal last : integer range 0 to 255;
  type data_array is array(0 to 255) of std_logic_vector(18 downto 0);
  type pos_array is array(0 to 255) of std_logic_vector(7 downto 0);
  signal data : data_array;
  signal position : pos_array;
  signal dout : std_logic_vector(18 downto 0);
  signal posout : std_logic_vector(7 downto 0);
begin
  process (clk, rst) begin
    if clk = '1' and clk'event then
      if rst = '1' or en = '0' then
        finish_insert <= '0';
        finish_sort <= '0';
        finish <= '0';
        counter <= 0;
        i <= 0;
        j <= 1;
        last <= to_integer(nbins) - 1;
        data <= (others => (others => '0'));
        position <= (others => (others => '0'));
        dout <= (others => '0');
        posout <= (others => '0');
      else
```
```
elsif finish_insert = '0' and counter < to_integer(nbins) then
  -- Inserting data to the datain array
  data(counter) <= data_in;
  position(counter) <= std_logic_vector(to_unsigned(counter, position(0)'length));
  counter <= counter + 1;
elsif finish_sort = '0' and last > 0 then
  -- Sorter parameters are entered
  if finish_insert = '0' then
    finish_insert <= '1';
    counter <= 0;
  end if;
  if order = '0' then
    if data(i) > data(j) then
      data(j) <= data(i);
      data(i) <= data(j);
      position(j) <= position(i);
      position(i) <= position(j);
    end if;
  end if;
  if order = '1' then
    if data(i) < data(j) then
      data(j) <= data(i);
      data(i) <= data(j);
      position(j) <= position(i);
      position(i) <= position(j);
    end if;
  end if;
i <= i + 1;
j <= j + 1;
if j = last then
  i <= 0;
j <= 1;
  last <= last - 1;
end if;
elseif finish = '0' and counter < 255 then
  -- Output the ordered data
  if finish_sort = '0' then
    finish_sort <= '1';
  end if;
dout <= data(counter);
posout <= position(counter);
counter <= counter + 1;
end if;
end process;
data_out <= dout;
pos_out <= posout;
end_sort <= finish_sort;
end_flag <= finish;
end architecture Behavioral;
B.15 Threshold Histogram Computation Wrapper

threshold_histogram_computation_wrapper.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity threshold_histogram_computation_wrapper is
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        valleys : in unsigned(1 downto 0);
        valley1 : in std_logic_vector(7 downto 0);
        valley2 : in std_logic_vector(7 downto 0);
        hist_data_in : in std_logic_vector(18 downto 0);
        grad_data_in : in std_logic_vector(7 downto 0);
        npixels : in unsigned(18 downto 0);
        TH1 : out std_logic_vector(18 downto 0);
        TH2 : out std_logic_vector(18 downto 0);
        TH3 : out std_logic_vector(18 downto 0);
        TM1 : out std_logic_vector(18 downto 0);
        TM2 : out std_logic_vector(18 downto 0);
        TM3 : out std_logic_vector(18 downto 0);
        TL1 : out std_logic_vector(18 downto 0);
        TL2 : out std_logic_vector(18 downto 0);
        TL3 : out std_logic_vector(18 downto 0);
        grad_addr_in : out std_logic_vector(17 downto 0);
        hist_addr_in : out std_logic_vector(7 downto 0);
        end_flag : out std_logic);
end threshold_histogram_computation_wrapper;

architecture wrapper of threshold_histogram_computation_wrapper is
  --- Components declaration ---
  --- Computation of the different parameters needed ---
  component params_computation
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          data_in : in std_logic_vector(18 downto 0);
          end_previous : in std_logic;
          mean : out unsigned(18 downto 0);
          median : out unsigned(18 downto 0);
          maximum : out unsigned(18 downto 0);
          minimum : out unsigned(18 downto 0);
          end_flag : out std_logic);
  end component;
  --- Threshold computation for the entry parameters ---
  component threshold_histogram_computation
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          mean : in unsigned(18 downto 0);
          median : in unsigned(18 downto 0);
```

130
maximum : in unsigned (18 downto 0);
minimum : in unsigned (18 downto 0);
grad_mean : in unsigned (18 downto 0);
grad_max : in unsigned (18 downto 0);
TH : out unsigned (18 downto 0);
TM : out unsigned (18 downto 0);
TL : out unsigned (18 downto 0);
end_flag : out std_logic);

end component;

-- Used to compute the gradient histogram --
component histogram_wrapper
  port (clk : in std_logic;
reset : in std_logic;
cntr_value : in std_logic_vector (18 downto 0);
pulse_start_input : in std_logic;
im_douta : in std_logic_vector (7 downto 0);
histo_dina : out std_logic_vector (18 downto 0);
histo_addr_a : out std_logic_vector (7 downto 0);
histo_wea : out std_logic_vector (0 downto 0);
histo_addr_b : out std_logic_vector (7 downto 0);
histo_doutb : in std_logic_vector (18 downto 0);
end_flag : out std_logic);
end component;

-- Memory where the gradient histogram is stored --
component histo_ram2
  port (clk_in : in std_logic;
clk_out : in std_logic;
we : in std_logic_vector (0 downto 0);
addr_in : in std_logic_vector (7 downto 0);
addr_out : in std_logic_vector (7 downto 0);
data_in : in std_logic_vector (18 downto 0);
data_out : out std_logic_vector (18 downto 0));
end component;

-- Signals declaration --
signal en_params2, en_params3 : std_logic;
signal mean1, median1, maximum1, minimum1 : unsigned (18 downto 0);
signal mean2, median2, maximum2, minimum2 : unsigned (18 downto 0);
signal mean3, median3, maximum3, minimum3 : unsigned (18 downto 0);
signal mean_grad, median_grad, maximum_grad, minimum_grad : unsigned (18 downto 0);
signal THH1, TM1, TLL1, THH2, TM2, TLL2, THH3, TM3, TLL3 : unsigned (18 downto 0);
signal start_histo, work_histo, started : std_logic;
signal histo_addr_a, histo_addr_b, histo_addr_b_histo : std_logic_vector (7 downto 0);
signal histo_dina, histo_doutb : std_logic_vector (18 downto 0);
begin

--- Port mapping ---

--- Params computation ---

params1 : params_computation
  port map(clk => clk, rst => rst, en => en, data_in => hist_data_in, end_previous => finish_valley1, mean => mean1, median => median1, maximum => maximum1, minimum => minimum1, end_flag => finish_params1);

params2 : params_computation
  port map(clk => clk, rst => rst, en => en_params2, data_in => hist_data_in, end_previous => finish_valley2, mean => mean2, median => median2, maximum => maximum2, minimum => minimum2, end_flag => finish_params2);

params3 : params_computation
  port map(clk => clk, rst => rst, en => en_params3, data_in => hist_data_in,
end_previous => finish_valley3,
mean   => mean3,
median => median3,
maximum=> maximum3,
minimum=> minimum3,
end_flag=> finish_params3);

-- Threshold computation --
thresh1: threshold_histogram_computation
port map(clk  => clk,
rst  => rst,
en   => enable_comp1,
mean => mean1,
median => median1,
maximum => maximum1,
minimum => minimum1,
grad_mean => mean_grad,
grad_max => maximum_grad,
TH => THH1,
TM => TMM1,
TL => TLL1,
end_flag  => finish_thresh1);

thresh2: threshold_histogram_computation
port map(clk  => clk,
rst  => rst,
en   => enable_comp2,
mean => mean2,
median => median2,
maximum => maximum2,
minimum => minimum2,
grad_mean => mean_grad,
grad_max => maximum_grad,
TH => THH2,
TM => TMM2,
TL => TLL2,
end_flag  => finish_thresh2);

thresh3: threshold_histogram_computation
port map(clk  => clk,
rst  => rst,
en   => enable_comp3,
mean => mean3,
median => median3,
maximum => maximum3,
minimum => minimum3,
grad_mean => mean_grad,
grad_max => maximum_grad,
TH => THH3,
TM => TMM3,
TL => TLL3,
end_flag  => finish_thresh3);
**Gradient histogram**

```vhdl
-- Gradient histogram --
nego : histogram_wrapper

port map(clk => clk,
reset => localrst,
ctrn_value => std_logic_vector(npixels),
pulse_start_input => start_histo,
im_douta => grad_data_in,
histo_dina => histo_dina,
histo_addra => histo_addra,
histo_wea => histo_wea,
histo_addrb => histo_addrb_histo,
histo_doutb => histo_doutb,
end_flag => finish_histo);

histo_ram : histo_ram2

port map(clk_in => clk,
clk_out => clk,
we => histo_wea,
addr_in => histo_addra,
addr_out => histo_addrb,
data_in => histo_dina,
data_out => histo_doutb);

params_grad : params_computation

port map(clk => clk,
rst => rst,
en => end_histo,
data_in => histo_doutb,
end_previous => finish_param,
mean => mean_grad,
median => median_grad,
maximum => maximum_grad,
minimum => minimum_grad,
end_flag => finish_params_grad);

**Process to scan all the addresses of the gradient histogram memory in order**

**to compute the gradient parameters**

addr_param : process(clk)

begin
  if(clk'event and clk = '1') then
    if (rst = '1' or en = '0') then
      histo_addrb_param <= "00000001";
      finish_param <= '0';
    elsif end_histo = '1' and histo_addrb_param < to_unsigned(254,
      histo_addrb_param'length) then
      histo_addrb_param <= histo_addrb_param + 1;
    elsif histo_addrb_param > to_unsigned(253, histo_addrb_param'length
      ) then
      finish_param <= '1';
    end if;
```
end if;
end process;

-- Process to compute the address of the image histogram

addr_histo : process(clk)
begin
if(clk'event and clk = '1') then
  if(rst = '1' or en = '0') then
    addr_hist <= (others => '0');
    finish_valley1 <= '0';
    finish_valley2 <= '0';
    finish_valley3 <= '0';
    en_params2 <= '0';
    en_params3 <= '0';
    elsif addr_hist < to_unsigned(255, addr_hist'length) then
      addr_hist <= addr_hist + 1;
      if (nvalleys > to_unsigned(0, nvalleys'length) and addr_hist =
          unsigned(valley1)) then
        finish_valley1 <= '1';
        en_params2 <= '1';
      end if;
      if (nvalleys > to_unsigned(1, nvalleys'length) and addr_hist =
          unsigned(valley2)) then
        finish_valley2 <= '1';
        en_params3 <= '1';
      end if;
    else
      if nvalleys = to_unsigned(0, nvalleys'length) then
        finish_valley1 <= '1';
      elsif nvalleys = to_unsigned(1, nvalleys'length) then
        finish_valley2 <= '1';
      else
        finish_valley3 <= '1';
      end if;
    end if;
  end if;
end process;

-- Process to compute the address of the image gradient

addr_gradient : process(clk)
begin
if(clk'event and clk = '1') then
  if(rst = '1' or en = '0') then
    addr_grad <= (others => '0');
    start_histo <= '0';
    started <= '0';
    end_histo <= '0';
    elsif start_histo = '0' and started = '0' then
      started <= '1';
      start_histo <= '1';
    elsif addr_grad < npixels and finish_histo = '0' then
      start_histo <= '0';
      addr_grad <= addr_grad + 1;
  end if;
end process;
else
    end_histo <= '1';
    addr_grad <= (others => '0');
end if;
end process;

fin : process(nvalleys, finish_thresh1, finish_thresh2, finish_thresh3)
begin
  if (nvalleys = "10" and finish_thresh1 = '1' and finish_thresh2 = '1' and finish_thresh3 = '1') then
    finish <= '1';
  elsif (nvalleys = "01" and finish_thresh1 = '1' and finish_thresh2 = '1') then
    finish <= '0';
  elsif (nvalleys = "00" and finish_thresh1 = '1') then
    finish <= '1';
  else
    finish <= '0';
  end if;
end process;

enable_comp1 <= finish_params1 and finish_params_grad;
enable_comp2 <= finish_params2 and finish_params_grad;
enable_comp3 <= finish_params3 and finish_params_grad;
work_histo <= en and not(end_histo);
localrst <= not(en);

histo_addrb <= histo_addrb_histo when work_histo = '1' else std_logic_vector(histo_addrb_param);
grad_addr_in <= std_logic_vector(addr_grad);
hist_addr_in <= std_logic_vector(addr_hist);

TH1 <= std_logic_vector(THH1);
TM1 <= std_logic_vector(TMM1);
TL1 <= std_logic_vector(TLL1);
TH2 <= std_logic_vector(THH2);
TM2 <= std_logic_vector(TMM2);
TL2 <= std_logic_vector(TLL2);
TH3 <= std_logic_vector(THH3);
TM3 <= std_logic_vector(TMM3);
TL3 <= std_logic_vector(TLL3);

end flag <= finish;

end wrapper;
B.16 Parameters Computation

params_computation.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity params_computation is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(18 downto 0);
         end_previous : in std_logic;
         mean : out unsigned(18 downto 0);
         median : out unsigned(18 downto 0);
         maximum : out unsigned(18 downto 0);
         minimum : out unsigned(18 downto 0);
         end_flag : out std_logic);
end params_computation;

architecture Behavioral of params_computation is
  -- Params computed
  -- Mean: will be computed when the previous block finishes
  -- Median: will be computed when the previous block finishes
  -- Max: will be computed everytime data gets in
  -- Min: will be computed everytime data gets in
  -- The maximum amount of bins that can be computed is 256
  -- Signals declaration
  type data_array is array(0 to 255) of unsigned(18 downto 0);
  signal localmean , localmedian , localmax , localmin : unsigned(18 downto 0);
  signal sumval : unsigned(18 downto 0);
  signal data : data_array;
  signal finish : std_logic;
  -- Bubble sorter signals
  signal i, last : integer range 0 to 255;
  signal j : integer range 1 to 255;
  signal medpos : unsigned(7 downto 0);

begin
  process (clk, rst)
  begin
    variable counter : integer range 0 to 255;
    begin
      if (clk'event and clk = '1') then
        if (rst = '1' or en = '0') then
          -- Code...
        end if;
      else
          -- Code...
      end if;
  end process;
```

137
localmean <= (others => '0');
localmax <= (others => '0');
localmin <= (others => '1');
sumval <= (others => '0');
data <= (others => (others => '0'));
finish <= '0';
counter := 0;
medpos <= (others => '0');
i <= 0;
j <= 1;
last <= 0;
elif end_previous = '0' then
  data(counter) <= unsigned(data_in);
  sumval <= sumval + unsigned(data_in);
  -- increment the counter to compute the mean and median later
  if counter < 255 then
    counter := counter + 1;
  end if;
  if localmax < unsigned(data_in) then
    localmax <= unsigned(data_in);
  end if;
  if localmin > unsigned(data_in) then
    localmin <= unsigned(data_in);
  end if;
  last <= counter-1;
else
  localmean <= sumval/counter;
  medpos <= to_unsigned(counter, medpos'length)/2;
  if last > 0 then
    if data(i) > data(j) then
      data(j) <= data(i);
      data(i) <= data(j);
    end if;
    i <= i+1;
    j <= j+1;
    if j = last then
      i <= 0;
      j <= 1;
      last <= last-1;
    end if;
  else
    finish <= '1';
  end if;
end if;
end process;
localmedian <= data(to_integer(medpos));
mean <= localmean;
median <= localmedian;
maximum <= localmax;
minimum <= localmin;
end_flag <= finish;
end architecture Behavioral;
B.17 Threshold Histogram Computation

```vhdl
library ieee;
library floatfixlib;
library work;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use floatfixlib.fixed_pkg.all;
use floatfixlib.math_utility_pkg.all;
use work.customized_operations_pkg.all;

entity threshold_histogram_computation is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         mean : in unsigned (18 downto 0);
         median : in unsigned (18 downto 0);
         maximum : in unsigned (18 downto 0);
         minimum : in unsigned (18 downto 0);
         grad_mean : in unsigned (18 downto 0);
         grad_max : in unsigned (18 downto 0);
         TH : out unsigned (18 downto 0);
         TM : out unsigned (18 downto 0);
         TL : out unsigned (18 downto 0);
         end_flag : out std_logic ) ;
end threshold_histogram_computation ;

architecture Behavioral of threshold_histogram_computation is
  -- Signals declarations
  signal finish : std_logic ;
  signal thl : ufixed (118 downto -79) ;
  signal thh : ufixed (29 downto -12) ;
  signal thm : ufixed (119 downto -81) ;
  signal RR1 : ufixed (9 downto -10) ;
  signal RR2 : ufixed (9 downto -10) ;
  begin
  process ( clk )
  variable R1 , R2 : ufixed (18 downto -19) := to_ufixed (0 , 18 , -19) ;
  variable R3 : ufixed (18 downto -19) ;
  variable R4 : ufixed (19 downto -20) ;
  variable RR1l : ufixed (57 downto -39) := to_ufixed (0 , 57 , -39) ;
  variable RR2l : ufixed (38 downto -20) := to_ufixed (0 , 38 , -20) ;
  variable R1a , R2a : ufixed (9 downto -10) ;
  variable R1b , R2b : ufixed (41 downto -22) ;
  variable R1c , R2c : ufixed (64 downto -43) ;
  variable R1d , R2d : ufixed (9 downto -10) ;
  variable thlvar : ufixed (118 downto -79) ;
  variable thhvar : ufixed (29 downto -12) ;
  begin
    if ( clk ' event and clk = ' 1 ' ) then
      if ( rst = ' 1 ' or en = ' 0 ' ) then
        ...
      end if ;
    end if ;
  end process ;
end Behavioral ;
```

thh <= (others => '0');

```
thm <= (others => '0');

thl <= (others => '0');

RR1 <= (others => '0');

RR2 <= (others => '0');

finish <= '0';
```

```
else

-- Common params
R1 := to_ufixed(mean, 18, 0)/to_ufixed(maximum, 18, 0);
R2 := to_ufixed(median, 18, 0)/to_ufixed(mean, 18, 0);

-- Lowest threshold computation
R3 := to_ufixed(grad_mean, 18, 0)/to_ufixed(grad_max, 18, 0);
R4 := (to_ufixed(maximum, 18, 0) - to_ufixed(minimum, 18, 0))/3;

```

```
RR1l := 1/(1+2*R1);
RR2l := 1/(1+R2);
thlvar := (RR1l*RR2l+R3)*R4+to_ufixed(minimum, 18, 0);

thl <= thlvar;

-- Highest threshold computation
R1a := ufixed_sqrt('0' & R1 & '0');
R2a := ufixed_sqrt('0' & R2 & '0');
R1b := (1/(1+R1a))*(1/(1+R1a));
R2b := (1/(1+R2a))*(1/(1+R2a));
R1c := 1/(1+R1b);
R2c := 1/(1+R2b);
R1d := ufixed_sqrt(R1c(19 downto -20));
R2d := ufixed_sqrt(R2c(19 downto -20));

```

```
thsvar := to_ufixed(0.5, -1, -2)*(R1d + R2d)*to_ufixed(maximum, 18, 0);
```

```
ths <= thsvar;

-- Medium threshold computation
thm := (thsvar + thlvar)*to_ufixed(0.5, -1, -2);

-- thl <= ((1 / (1 + 2 * mean / maximum))) * (1 / (1 + median / mean)) * maximum / 3;

```

```
-- thh <= 1 / 2 * ((1 / (1 + (1 / (1 + sqrt(mean / maximum)))) ** 2)) ** (1 / 2) + (1 / (1 + (1 / (1 + sqrt(median / mean)))) ** 2)) * maximum;

```

```
-- thm <= (tl + th) / 2;

```

```
finish <= '1';

end if;
end if;
end process;
```

```
TH <= unsigned(thh(18 downto 0));
TM <= unsigned(thm(18 downto 0));
TL <= unsigned(thl(18 downto 0));
end_flag <= finish;
```

end architecture Behavioral;
B.18 Histogram Thresholding Wrapper

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity histogram_thresholding_wrapper is
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        valleys : in unsigned(1 downto 0);
        valley1 : in std_logic_vector(7 downto 0);
        valley2 : in std_logic_vector(7 downto 0);
        TH1 : in std_logic_vector(18 downto 0);
        TH2 : in std_logic_vector(18 downto 0);
        TH3 : in std_logic_vector(18 downto 0);
        TM1 : in std_logic_vector(18 downto 0);
        TM2 : in std_logic_vector(18 downto 0);
        TM3 : in std_logic_vector(18 downto 0);
        TL1 : in std_logic_vector(18 downto 0);
        TL2 : in std_logic_vector(18 downto 0);
        TL3 : in std_logic_vector(18 downto 0);
        hist_data_in : in std_logic_vector(18 downto 0);
        edge_bin_high : out std_logic_vector(255 downto 0);
        edge_bin_mid : out std_logic_vector(255 downto 0);
        edge_bin_low : out std_logic_vector(255 downto 0);
        hist_addr_in : out std_logic_vector(7 downto 0);
        end_flag : out std_logic);
end histogram_thresholding_wrapper;

architecture wrapper of histogram_thresholding_wrapper is
  component histogram_thresholding
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        th : in unsigned(18 downto 0);
        tm : in unsigned(18 downto 0);
        tl : in unsigned(18 downto 0);
        hist_data_in : in std_logic_vector(18 downto 0);
        addr_start : in std_logic_vector(7 downto 0);
        addr_stop : in std_logic_vector(7 downto 0);
        edge_bin_high : out std_logic_vector(255 downto 0);
        edge_bin_mid : out std_logic_vector(255 downto 0);
        edge_bin_low : out std_logic_vector(255 downto 0);
        hist_addr_in : out std_logic_vector(7 downto 0);
        end_flag : out std_logic);
  end component;

  — Signals
  signal edge_bin_low1, edge_bin_mid1, edge_bin_high1 :
    std_logic_vector(255 downto 0);
```

142
signal edge_bin_low2, edge_bin_mid2, edge_bin_high2 : std_logic_vector(255 downto 0);
signal edge_bin_low3, edge_bin_mid3, edge_bin_high3 : std_logic_vector(255 downto 0);
signal addr1, addr2, addr3 : std_logic_vector(7 downto 0);
signal end_thresh1, end_thresh2, end_thresh3 : std_logic;
signal en_thresh2, en_thresh3 : std_logic;
signal work_thresh1, work_thresh2, work_thresh3 : std_logic;
signal finish : std_logic;

begin
histo_thresh1 : histogram_thresholding
  port map(clk => clk,
            rst => rst,
            en => en,
            th => unsigned(TH1),
            tm => unsigned(TM1),
            tl => unsigned(TL1),
            hist_data_in => hist_data_in,
            addr_start => "00000000",
            addr_stop => valley1,
            edge_bin_high => edge_bin_high1,
            edge_bin_mid => edge_bin_mid1,
            edge_bin_low => edge_bin_low1,
            hist_addr_in => addr1,
            end_flag => end_thresh1);

histo_thresh2 : histogram_thresholding
  port map(clk => clk,
            rst => rst,
            en => en_thresh2,
            th => unsigned(TH2),
            tm => unsigned(TM2),
            tl => unsigned(TL2),
            hist_data_in => hist_data_in,
            addr_start => valley1,
            addr_stop => valley2,
            edge_bin_high => edge_bin_high2,
            edge_bin_mid => edge_bin_mid2,
            edge_bin_low => edge_bin_low2,
            hist_addr_in => addr2,
            end_flag => end_thresh2);

histo_thresh3 : histogram_thresholding
  port map(clk => clk,
            rst => rst,
            en => en_thresh3,
            th => unsigned(TH3),
            tm => unsigned(TM3),
            tl => unsigned(TL3),
            hist_data_in => hist_data_in,
            addr_start => valley2,
addr_stop  ⇒ "11111111",
edge_bin_high  ⇒ edge_bin_high3,
edge_bin_mid  ⇒ edge_bin_mid3,
edge_bin_low  ⇒ edge_bin_low3,
hist_addr_in  ⇒ addr3,
end_flag  ⇒ end_thresh3
);

mx : process ( end_thresh1, end_thresh2, end_thresh3, nvalleys, finish )
begin
  if end_thresh3 = '1' then
    finish <= '1';
  elsif end_thresh2 = '1' then
    if nvalleys = "01" then
      finish <= '1';
    else
      en_thresh3 <= '1';
    end if;
  elsif end_thresh1 = '1' then
    if nvalleys = "00" then
      finish <= '1';
    else
      en_thresh2 <= '1';
    end if;
  else
    finish <= '0';
    en_thresh2 <= '0';
    en_thresh3 <= '0';
  end if;
end process;

work_thresh1 <= en and ( not end_thresh1);
work_thresh2 <= en_thresh2 and ( not end_thresh2);
work_thresh3 <= en_thresh3 and ( not end_thresh3);
edge_bin_low  <= edge_bin_low1 or edge_bin_low2 or edge_bin_low3;
edge_bin_mid  <= edge_bin_mid1 or edge_bin_mid2 or edge_bin_mid3;
edge_bin_high  <= edge_bin_high1 or edge_bin_high2 or edge_bin_high3;
hist_addr_in  <= addr1 when work_thresh1 = '1' else
  addr2 when work_thresh2 = '1' else
  addr3 when work_thresh3 = '1' else
  ( others => '0' );
end_flag <= finish;
end architecture wrapper;
B.19 Histogram Thresholding

histogram_thresholding.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

-- sobel kernel will compute the sobel gradient of the tile 3x3 in 1 clock
entity histogram_thresholding is
  port (clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         th : in unsigned(18 downto 0); -- High threshold
         tm : in unsigned(18 downto 0); -- Medium threshold
         tl : in unsigned(18 downto 0); -- Low threshold
         hist_data_in : in std_logic_vector(18 downto 0);
         addr_start : in std_logic_vector(7 downto 0);
         addr_stop : in std_logic_vector(7 downto 0);
         ― Vectors with edge positions = 1 and flat = 0
         edge_bin_high : out std_logic_vector(255 downto 0);
         edge_bin_mid : out std_logic_vector(255 downto 0);
         edge_bin_low : out std_logic_vector(255 downto 0);
         hist_addr_in : out std_logic_vector(7 downto 0);
         end_flag : out std_logic);
end histogram_thresholding;

architecture Behavioral of histogram_thresholding is
  ― Signals declarations
  signal bin_high , bin_mid , bin_low : std_logic_vector(255 downto 0);
  signal value_high, value_mid, value_low : std_logic;
  signal hist_addr, old_hist_addr : unsigned(7 downto 0);
  signal started, finish, prefinish : std_logic;

begin
  process(clk, rst)
  begin
    if (clk'event and clk = '1') then
      if (rst = '1' or en = '0') then -- restart all the procedure
        bin_high <= (others => '0');
        bin_mid <= (others => '0');
        bin_low <= (others => '0');
        hist_addr <= (others => '0');
        value_high <= '0';
        value_mid <= '0';
        value_low <= '0';
        started <= '0';
        finish <= '0';
        prefinish <= '0';
      elsif (en = '1' and started = '0') then
```
started <= '1';
hist_addr <= unsigned(addr_start);

elsif finish = '0' then
  old_hist_addr <= hist_addr;
  hist_addr <= hist_addr + 1;
  if th < unsigned(hist_data_in) then
    value_high <= '0';
  else
    value_high <= '1';
  end if;
  if tm < unsigned(hist_data_in) then
    value_mid <= '0';
  else
    value_mid <= '1';
  end if;
  if tl < unsigned(hist_data_in) then
    value_low <= '0';
  else
    value_low <= '1';
  end if;
  bin_high(to_integer(old_hist_addr)) <= value_high;
  bin_mid(to_integer(old_hist_addr)) <= value_mid;
  bin_low(to_integer(old_hist_addr)) <= value_low;
  if hist_addr = unsigned(addr_stop) then
    prefinish <= '1';
  end if;
  if prefinish = '1' then
    finish <= '1';
  end if;
end if;

hist_addr_in <= std_logic_vector(hist_addr);
edge_bin_high <= bin_high;
edge_bin_mid <= bin_mid;
edge_bin_low <= bin_low;
end_flag <= finish;

end Behavioral;
B.20 Pixel Grouping Wrapper

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--- Pixel grouping wrapper ---
-- Groups 3 pixel grouping blocks in one to have a block with grouping for all
-- thresholds
entity pixel_grouping_wrapper is
port (clk : in std_logic;
rst : in std_logic;
en : in std_logic;
data_im : in std_logic_vector(7 downto 0);
edge_bin_low : in std_logic_vector(255 downto 0);
edge_bin_mid : in std_logic_vector(255 downto 0);
edge_bin_high : in std_logic_vector(255 downto 0);
gr_data_in : in std_logic_vector(7 downto 0);
npixels : in unsigned(18 downto 0);
gr_edge_low : out std_logic_vector(7 downto 0);
gr_flat_low : out std_logic_vector(7 downto 0);
gr_edge_mid : out std_logic_vector(7 downto 0);
gr_flat_mid : out std_logic_vector(7 downto 0);
gr_edge_high : out std_logic_vector(7 downto 0);
gr_flat_high : out std_logic_vector(7 downto 0);
npixels_edge_low : out unsigned(18 downto 0);
npixels_flat_low : out unsigned(18 downto 0);
npixels_edge_mid : out unsigned(18 downto 0);
npixels_flat_mid : out unsigned(18 downto 0);
npixels_edge_high : out unsigned(18 downto 0);
npixels_flat_high : out unsigned(18 downto 0);
addr_im : out std_logic_vector(17 downto 0);
gr_addr_in : out std_logic_vector(17 downto 0);
addr_mask : out std_logic_vector(17 downto 0);
we : out std_logic;
end_flag : out std_logic);
end pixel_grouping_wrapper;

architecture wrapper of pixel_grouping_wrapper is

--- Components declaration ---

component pixel_grouping
port (clk : in std_logic;
rst : in std_logic;
en : in std_logic;
data_im : in std_logic_vector(7 downto 0);
edge_bin : in std_logic_vector(255 downto 0);
gr_data_in : in std_logic_vector(7 downto 0);
end pixel_grouping;
```

147
begin

--- Port mapping ---

pixel_group_low : pixel_grouping

  port map(
    clk => clk,
    rst => rst,
    en => en,
    data_im => data_im,
    edge_bin => edge_bin_low,
    grad_data_in => grad_data_in,
    npixels => npixels,
    grad_edge => grad_edge_low,
    grad_flat => grad_flat_low,
    npixels_edge => npixels_edge_low,
    npixels_flat => npixels_flat_low,
    addr_im => addr_im,
    grad_addr_in => grad_addr_in,
    addr_mask => addr_mask,
    we => we,
    end_flag => end_flag
  );

pixel_group_mid : pixel_grouping

  port map(
    clk => clk,
    rst => rst,
    en => en,
    data_im => data_im,
    edge_bin => edge_bin_mid,
    grad_data_in => grad_data_in,
    npixels => npixels,
    grad_edge => grad_edge_mid,
    grad_flat => grad_flat_mid,
    npixels_edge => npixels_edge_mid,
    npixels_flat => npixels_flat_mid,
    addr_im => addr_im,
    grad_addr_in => grad_addr_in,
    addr_mask => addr_mask,
    we => we,
    end_flag => end_flag
  );

pixel_group_high : pixel_grouping

end component;
port map(clk => clk,
    rst => rst,
    en => en,
    data_im => data_im,
    edge_bin => edge_bin_high,
    grad_data_in => grad_data_in,
    npixels => npixels,
    grad_edge => grad_edge_high,
    grad_flat => grad_flat_high,
    npixels_edge => npixels_edge_high,
    npixels_flat => npixels_flat_high,
    addr_im => addr_im,
    grad_addr_in => grad_addr_in,
    addr_mask => addr_mask,
    we => we,
    end_flag => end_flag
);
end wrapper;
B.21  Pixel Grouping

/* pixel_grouping.vhd */
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
/* sobel kernel will compute the sobel gradient of the tile 3x3 in 1 clock */

entity pixel_grouping is
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        data_im : in std_logic_vector(7 downto 0);
        edge_bin : in std_logic_vector(255 downto 0);
        grad_data_in : in std_logic_vector(7 downto 0);
        npixels : in unsigned(18 downto 0);
        grad_edge : out std_logic_vector(7 downto 0);
        grad_flat : out std_logic_vector(7 downto 0);
        npixels_edge : out unsigned(18 downto 0);
        npixels_flat : out unsigned(18 downto 0);
        addr_im : out std_logic_vector(17 downto 0);
        addr_in : out std_logic_vector(17 downto 0);
        addr_mask : out std_logic_vector(17 downto 0);
        addr : out std_logic);
  end pixel_grouping;
architecture Behavioral of pixel_grouping is
  begin
    process (clk, rst)
    begin
      if (clk'event and clk = '1') then
        if (rst = '1' or en = '0') then  -- restart all the procedure
          addr_in <= (others => '0');
          addr_out <= (others => '0');
          npixels_edge <= (others => '0');
          npixels_flat <= (others => '0');
          started <= '0';
          finish <= '0';
          we <= '0';
          elsif (en = '1' and started = '0') then
            started <= '1';
        elsif addr_in < npixels then
          begin
            addr_in <= addr_in + 1;
          end
        end if;
addr_in <= addr_in + 1;
addr_out <= addr_in;
we <= '1';
if edge_bin(to_integer(unsigned(data_im))) = '1' then
    npix_edge <= npix_edge + 1;
else
    npix_flat <= npix_flat + 1;
end if;
elsif finish = '0' then
    addr_out <= addr_in;
    we <= '0';
    finish <= '1';
end if;
ext if;
end process;

grad_edge <= grad_data_in when (edge_bin(to_integer(unsigned(data_im))) = '1') else
    (others => '0');
grad_flat <= grad_data_in when (edge_bin(to_integer(unsigned(data_im))) = '0') else
    (others => '0');
addr_im <= std_logic_vector(addr_in);
grad_addr_in <= std_logic_vector(addr_in);
addr_mask <= std_logic_vector(addr_out);
npixels_edge <= npix_edge;
npixels_flat <= npix_flat;
end_flag <= finish;
end Behavioral;
B.22 Classification

classification.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity classification is
  port ( clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in_edge : in std_logic_vector(7 downto 0);
         data_in_flat : in std_logic_vector(7 downto 0);
         npixels : in unsigned(18 downto 0);
         npixels_edge : in unsigned(18 downto 0);
         npixels_flat : in unsigned(18 downto 0);
         addr_in : out std_logic_vector(17 downto 0);
         data_out : out std_logic_vector(0 downto 0);
         addr_out : out std_logic_vector(17 downto 0);
         we : out std_logic;
         end_flag : out std_logic ) ;
end classification ;

architecture wrapper of classification is
  -- Components declaration
  component threshold_computation_edge_wrapper
    port ( clk : in std_logic;
           rst : in std_logic;
           en : in std_logic;
           grad_in : in std_logic_vector(7 downto 0);
           npixels : in unsigned(18 downto 0);
           tl : out std_logic_vector(7 downto 0);
           end_flag : out std_logic ) ;
  end component;

  component threshold_computation_flat_wrapper
    port ( clk : in std_logic;
           rst : in std_logic;
           en : in std_logic;
           grad_in : in std_logic_vector(7 downto 0);
           npixels : in unsigned(18 downto 0);
           npixels_flat : in unsigned(18 downto 0);
           th : out std_logic_vector(7 downto 0);
           end_flag : out std_logic ) ;
  end component;

  component comparison_binarization
    port ( data_in : in std_logic_vector(7 downto 0);
           thresh : in std_logic_vector(7 downto 0);
           data_out : out std_logic_vector(0 downto 0)) ;
  end component;

  -- Signal declarations
signal end_edge, end_flat, end_thresh, end_bin, start_comp :
  std_logic;
signal tl, th :
  std_logic_vector(7 downto 0);signal addr_data_in : unsigned (17 downto 0);
signal binary_edge, binary_flat :
  std_logic_vector(0 downto 0);

begin
  thresh_edge : threshold_computation_edge_wrapper
    port map(clk => clk,
             rst => rst,
             en => en,
             grad_in => data_in_edge,
             npixels => npixels,
             tl => tl,
             end_flag => end_edge);
  thresh_flat : threshold_computation_flat_wrapper
    port map(clk => clk,
             rst => rst,
             en => en,
             grad_in => data_in_flat,
             npixels => npixels,
             npixels_flat => npixels_flat,
             th => th,
             end_flag => end_flat);
  comp_edge : comparison_binarization
    port map(data_in => data_in_edge,
              thresh => tl,
              data_out => binary_edge);
  comp_flat : comparison_binarization
    port map(data_in => data_in_flat,
              thresh => th,
              data_out => binary_flat);

class_proc : process(clk, rst, en, end_thresh)
begin
  if (clk'event and clk = '1') then
    if (rst = '1' or en = '0') then
      addr_data_in <= (others => '0');
      end_bin <= '0';
      start_comp <= '0';
      elsif (end_thresh = '1' and start_comp = '0') then
        addr_data_in <= (others => '0');
        start_comp <= '1';
      elsif (end_thresh = '1' and addr_data_in = npixels(17 downto 0)
               and end_bin = '0') then
        end_bin <= '1';
      elsif end_bin = '0' then

addr.data_in <= addr.data_in + 1;
end if;
end if;
end process;
end thresh <= end_edge and end_flat;
end_flag <= end_bin;
addr.in <= std.logic_vector(addr.data_in);
addr.out <= std.logic_vector(addr.data_in);
we <= end_thresh and (not end_bin);
data_out <= binary_edge and binary_flat;
end wrapper;
B.23 Comparison and Binarization

comparison_binarization.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity comparison_binarization is
  port(data_in : in std_logic_vector(7 downto 0);
       thresh : in std_logic_vector(7 downto 0);
       data_out : out std_logic_vector(0 downto 0)
  );
end comparison_binarization;

architecture Behavioral of comparison_binarization is
begin
  process (data_in, thresh)
  begin
    if data_in < thresh then
      data_out(0) <= '1';
    else
      data_out(0) <= '0';
    end if;
  end process;
end Behavioral;
B.24 Threshold Computation Edge Wrapper

---

threshold_computation_edge_wrapper.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity threshold_computation_edge_wrapper is
  port (clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        grad_in : in std_logic_vector(7 downto 0);
        n_pixels : in unsigned(18 downto 0);
        tl : out std_logic_vector(7 downto 0);
        end_flag : out std_logic);
end threshold_computation_edge_wrapper;

architecture wrapper of threshold_computation_edge_wrapper is
  — Components declaration
  component histogram_wrapper
    port (clk : in std_logic;
          reset : in std_logic;
          cntr_value : in std_logic_vector(18 downto 0);
          pulse_start_input : in std_logic;
          im_douta : in std_logic_vector(7 downto 0);
          histo_dina : out std_logic_vector(18 downto 0);
          histo_addra : out std_logic_vector(7 downto 0);
          histo_wea : out std_logic_vector(0 downto 0);
          histo_addrb : out std_logic_vector(7 downto 0);
          histo_doutb : in std_logic_vector(18 downto 0);
          end_flag : out std_logic);
  end component;
  component histo_ram2
    port (clk_in : in std_logic;
          clk_out : in std_logic;
          we : in std_logic_vector(0 downto 0);
          addr_in : in std_logic_vector(7 downto 0);
          addr_out : in std_logic_vector(7 downto 0);
          data_in : in std_logic_vector(18 downto 0);
          data_out : out std_logic_vector(18 downto 0));
  end component;
  component hist_lowpass_filter is
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          data_in : in std_logic_vector(18 downto 0);
          addr_in : out std_logic_vector(7 downto 0);
          addr_out : out std_logic_vector(7 downto 0);
          data_out : out std_logic_vector(18 downto 0);
          we : out std_logic_vector(0 downto 0);
          end_flag : out std_logic);
  end component;
  component threshold_computation_edge
```
port (clk : in std_logic;
    rst : in std_logic;
    en : in std_logic;
    data_in : in std_logic_vector(18 downto 0);
    addr_in : in std_logic_vector(7 downto 0);
    tl : out std_logic_vector(7 downto 0));
end component;

-- Signal declarations
signal start_histo, end_histo, end_thresh, end_filt, localrst :
    std_logic;
signal histo_dina, histo_doutb :
    std_logic_vector(18 downto 0);
signal histo_addra, histo_addrb, addrb_hist :
    std_logic_vector(7 downto 0);
signal histo_wea, histo_wea_hist, we_filt :
    std_logic_vector(0 downto 0);
signal addrb_thresh :
    unsigned(7 downto 0);
signal addrb_filt, addra_filt, histo_addra_hist :
    std_logic_vector(7 downto 0);
signal work_histo, work_thresh, work_filt :
    std_logic;
signal histo_dina_filt, histo_dina_hist :
    std_logic_vector(18 downto 0);
begin
    histo_wrapper : histogram_wrapper
        port map(clk => clk,
                  reset => localrst,
                  cntr_value => std_logic_vector(npixels),
                  pulse_start_input => start_histo,
                  im_douta => grad_in,
                  histo_dina => histo_dina_hist,
                  histo_addra => histo_addra_hist,
                  histo_wea => histo_wea_hist,
                  histo_addrb => addrb_hist,
                  histo_doutb => histo_doutb,
                  end_flag => end_histo);
    histo_ram : histo_ram2
        port map(clk_in => clk,
                  clk_out => clk,
                  we => histo_wea,
                  addr_in => histo_addra,
                  addr_out => histo_addrb,
                  data_in => histo_dina,
                  data_out => histo_doutb);
    histo_filter : hist_lowpass_filter
        port map(clk => clk,
                 rst => rst,
en    => end_histo,  
data_in => histo_doutb,  
addr_in => addrb_filt,  
addr_out => addra_filt,  
data_out => histo_dina_filt,  
we    => we_filt,  
end_flag => end_filt  
);
thresh_edge : threshold_computation_edge  
    port map(clk => clk,  
rst  => rst,  
en   => end_filt,  
data_in => histo_doutb,  
addr_in => std_logic_vector(addrb_thresh),  
t1    => t1);

process(clk, en, rst)
begin  
  if (clk'event and clk = '1') then  
    if (rst = '1' or en = '0') then     --restart all the procedure  
      start_histo <= '0';  
    elsif (en'event and en = '1') then  
      start_histo <= '1';  
    elsif start_histo = '1' then  
      start_histo <= '0';  
    end if;  
  end if;  
end process;

process(clk, rst, end_filt)
begin  
  if (clk'event and clk = '1') then  
    if (rst = '1' or end_filt = '0') then  
      end_thresh <= '0';  
      addrb_thresh <= "00000001";  
    elsif (end_filt = '1' and addrb_thresh < "1111111") then  
      addrb_thresh <= addrb_thresh + 1;  
    else  
      end_thresh <= '1';  
    end if;  
  end if;  
end process;

localrst <= not(en);  
work_histo <= en and (not end_histo);  
work_filt <= end_histo and (not end_filt);  
work_thresh <= end_filt and (not end_thresh);  
end_flag <= end_thresh;  
histo_addrb <= addrb_hist when work_histo = '1' else  
          addrb_filt when work_filt = '1'  
else
\begin{verbatim}
  std_logic_vector(addrb_thresh) when work_thresh = '1'
    else
      others => '0');
  histo_addra <= histo_addra_histo when work_histo = '1' else
    addra_filt when work_filt = '1' else
      others => '0');
  histo_dina <= histo_dina_histo when work_histo = '1' else
    histo_dina_filt when work_filt = '1' else
      others => '0');
  histo_wea <= histo_wea_histo when work_histo = '1' else
    we_filt when work_filt = '1' else
      others => '0');
end wrapper;
\end{verbatim}


B.25 Threshold Computation Flat Wrapper

threshold_computation_flat_wrapper.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity threshold_computation_flat_wrapper is
  port (clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         grad_in : in std_logic_vector(7 downto 0);
         npixels : in unsigned(18 downto 0);
         npixels_flat : in unsigned(18 downto 0);
         th : out std_logic_vector(7 downto 0);
         end_flag : out std_logic)
  );
end threshold_computation_flat_wrapper;

architecture wrapper of threshold_computation_flat_wrapper is
  -- Components declaration
  component histogram_wrapper
    port (clk : in std_logic;
          reset : in std_logic;
          cntr_value : in std_logic_vector(18 downto 0);
          pulse_start_input : in std_logic;
          im_douta : in std_logic_vector(7 downto 0);
          histo_dina : out std_logic_vector(18 downto 0);
          histo_addra : out std_logic_vector(7 downto 0);
          histo_wea : out std_logic_vector(0 downto 0);
          histo_addrb : out std_logic_vector(7 downto 0);
          histo_doutb : in std_logic_vector(18 downto 0);
          end_flag : out std_logic);
  end component;

  component histo_ram2
    port (clk, in : in std_logic;
          clk_out : in std_logic;
          we : in std_logic_vector(0 downto 0);
          addr_in : in std_logic_vector(7 downto 0);
          addr_out : in std_logic_vector(7 downto 0);
          data_in : in std_logic_vector(18 downto 0);
          data_out : out std_logic_vector(18 downto 0));
  end component;

  component hist_lowpass_filter is
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          data_in : in std_logic_vector(18 downto 0);
          addr_in : out std_logic_vector(7 downto 0);
          addr_out : out std_logic_vector(7 downto 0);
          data_out : out std_logic_vector(18 downto 0);
          we : out std_logic_vector(0 downto 0);
          end_flag : out std_logic);
  end component;
```

160
component threshold_computation_flat
port (clk : in std_logic;
   rst : in std_logic;
   en : in std_logic;
   data_in : in std_logic_vector(18 downto 0);
   addr_in : in std_logic_vector(7 downto 0);
   npixels : in unsigned(18 downto 0);
   th : out std_logic_vector(7 downto 0));
end component;

-- Signal declarations
signal start_histo, end_histo, end_thresh, end_filt, localrst :
   std_logic;
signal histo_dina, histo_doutb :
   std_logic_vector(18 downto 0);
signal histo_addra, histo_addrb, addr_hist :
   std_logic_vector(7 downto 0);
signal histo_wea, histo_wea_histo, we_filt :
   std_logic_vector(0 downto 0);
signal addrb_thresh :
   unsigned(7 downto 0);
signal addrb_filt, addra_filt, histo_addra_hist :
   std_logic_vector(7 downto 0);
signal work_histo, work_thresh, work_filt :
   std_logic;
signal histo_dina_filt, histo_dina_histo :
   std_logic_vector(18 downto 0);
signal npix_flat :
   unsigned(18 downto 0);
begin
histo_wrapper : histogram_wrapper
   port map(clk => clk,
            reset => localrst,
            cntr_value => std_logic_vector(npixels),
            pulse_start_input => start_histo,
            im_douta => grad_in,
            histo_dina => histo_dina_histo,
            histo_addra => histo_addra_hist,
            histo_wea => histo_wea_histo,
            histo_addrb => addrb_hist,
            histo_doutb => histo_doutb,
            end_flag => end_histo
   );
histo_ram : histo_ram2
   port map(clk_in => clk,
            clk_out => clk,
            we => histo_wea,
            addr_in => histo_addra,
            addr_out => histo_addrb,
            data_in => histo_dina,
            data_out => histo_doutb
   );
histo_filter : hist_lowpass_filter  
port map(clk => clk,  
srst => rst,  
en => end_histo,  
data_in => histo_doutb,  
addr_in => addrb_filt,  
addr_out => addra_filt,  
data_out => histo_dina_filt,  
we => we_filt,  
end_flag => end_filt  
);

thresh_flat : threshold_computation_flat  
port map(clk => clk,  
srst => rst,  
en => end_filt,  
data_in => histo_doutb,  
addr_in => std_logic_vector(addrb_thresh),  
npixels => npix_flat,  
th => th);

process(clk, en, rst)  
begin  
if (clk'event and clk = '1') then  
if (rst = '1' or en = '0') then  
--restart all the procedure  
start_histo <= '0';  
elif (en'event and en = '1') then  
start_histo <= '1';  
elif start_histo = '1' then  
start_histo <= '0';  
end if;  
end if;  
end process;

process(clk, rst, end_filt)  
begin  
if(clk'event and clk = '1') then  
if (rst = '1' or end_filt = '0') then  
end_thresh <= '0';  
addrb_thresh <= "11111110";  
elif (end_filt = '1' and addrb_thresh > "00000001") then  
addrb_thresh <= addrb_thresh - 1;  
else  
end_thresh <= '1';  
end if;  
end if;  
end process;

process(rst, en, work_histo, grad_in)  
begin  
if (rst = '1' or en = '0') then  
npix_flat <= (others => '0');
elsif (work_histo = '1') then
    if (unsigned(grad_in) > "00000000" and unsigned(grad_in) < 
        "11111111") then
        npix_flat <= npix_flat + 1;
    end if;
end if;
end process;
loralrst <= not(en);
work_histo <= en and (not end_histo);
work_filt <= end_histo and (not end_filt);
work_thresh <= end_filt and (not end_thresh);
end_flag <= end_thresh;
histo_addrb <= addrb_hist when work_histo = '1' else
    addrb_filt when work_filt = '1'
else
    std_logic_vector(addrb_thresh) when work_thresh = '1'
    else
        (others => '0');
end
histo_addra <= histo_addra_hist when work_histo = '1' else
    addra_filt when work_filt = '1' else
        (others => '0');
histo_dina <= histo_dina_hist when work_histo = '1' else
    histo_dina_filt when work_filt = '1' else
        (others => '0');
histo_wea <= histo_wea_hist when work_histo = '1' else
    we_filt when work_filt = '1' else
        (others => '0');
end wrapper;
B.26 Threshold Computation Edge

threshold_computation_edge.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity threshold_computation_edge is
    port (clk : in std_logic;
          rst : in std_logic;
          en : in std_logic;
          data_in : in std_logic_vector(18 downto 0);
          addr_in : in std_logic_vector(7 downto 0);
          tl : out std_logic_vector(7 downto 0)
    );
end threshold_computation_edge;

architecture Behavioral of threshold_computation_edge is
    signal data0, data1, data2, data3 : std_logic_vector(18 downto 0);
    signal addr_prev : std_logic_vector(7 downto 0);
begin
    process(clk, rst, en)
        variable counter : integer range 0 to 4;
        variable minstored : signed(19 downto 0);  -- Provisional min value
        variable subs : signed(19 downto 0);  -- Gradient of hist (3 pos shift)
    begin
        if (clk'event and clk = '1') then
            if (rst = '1' or en = '0') then  -- restart all the procedure
                data0 <= (others => '0');
                data1 <= (others => '0');
                data2 <= (others => '0');
                data3 <= (others => '0');
                tl <= (others => '0');
                addr_prev <= (others => '0');
                counter := 0;
                minstored := (others => '1');
                minstored(19) := '0';  -- Signed signal (reset= all ones
                                       -- except the first bit)
                subs := (others => '1');
                subs(19) := '0';
            else
                subs := signed('0' & data0) - signed('0' & data3);
                if counter < 4 then
                    counter := counter + 1;
                elsif subs < minstored then
                    minstored := subs;
                    tl <= addr_prev;
                end if;
                data0 <= data_in;
                data1 <= data0;
            end if;
        end if;
    end process;
end Behavioral;
```
data2 <= data1;
data3 <= data2;
addr_prev <= addr_in;
end if;
end if;
end process;

end Behavioral;
B.27 Threshold Computation Flat

threshold_computation_flat.vhd

1 library ieee;
2 library floatfixlib;
3 use ieee.std_logic_1164.all;
4 use ieee.numeric.std.all;
5 use ieee.std_logic_unsigned.all;
6 use floatfixlib.fixed_pkg.all;
7 use floatfixlib.math_utility_pkg.all;
8
9 entity threshold_computation_flat is
10 port (clk : in std_logic;
11 rst : in std_logic;
12 en : in std_logic;
13 data_in : in std_logic_vector(18 downto 0);
14 addr_in : in std_logic_vector(7 downto 0);
15 npixels : in unsigned(18 downto 0);
16 th : out std_logic_vector(7 downto 0)
17 );
18 end threshold_computation_flat;
19
20 architecture Behavioral of threshold_computation_flat is
21 signal data0, data1, data2, data3 : std_logic_vector(18 downto 0);
22 signal addr_prev : std_logic_vector(7 downto 0);
23 signal finish : std_logic;
24 begin
25 process (clk, rst, en)
26 variable counter : integer range 0 to 4;
27 variable subs : signed(19 downto 0); — Gradient of hist (3 pos shift)
28 variable thresh : ufixed(18 downto -100);
29 begin
30 if (clk'event and clk = '1') then
31 if (rst = '1' or en = '0') then — restart all the procedure
32 data0 <= (others => '0');
33 data1 <= (others => '0');
34 data2 <= (others => '0');
35 data3 <= (others => '0');
36 th <= (others => '0');
37 finish <= '0';
38 counter := 0;
39 subs := (others => '0');
40 thresh := to_ufixed(npixels, 18, 0)*to_ufixed(0.001, -1, -100);
41 else
42 subs := abs(signed('0' & data0) - signed('0' & data3));
43 if counter < 4 then
44 counter := counter + 1;
45 elsif unsigned(subs(18 downto 0)) < unsigned(thresh(18 downto 0)) and finish = '0' then
46 th <= addr_prev;
47 end process;

elsif finish = '0' then
    finish <= '1';
end if;
data0 <= data_in;
data1 <= data0;
data2 <= data1;
data3 <= data2;
addr_prev <= addr_in;
end if;
end if;
end process;
end Behavioral;
B.28 Filter Kernel

filterkernel.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity filterkernel is
port (clk : in std_logic;
      rst : in std_logic;
      en : in std_logic;
      start : in std_logic;
      p1 : in std_logic_vector(7 downto 0);
      p2 : in std_logic_vector(7 downto 0);
      p3 : in std_logic_vector(7 downto 0);
      p4 : in std_logic_vector(7 downto 0);
      p5 : in std_logic_vector(7 downto 0);
      p6 : in std_logic_vector(7 downto 0);
      p7 : in std_logic_vector(7 downto 0);
      p8 : in std_logic_vector(7 downto 0);
      p9 : in std_logic_vector(7 downto 0);
      mask_val : in std_logic;
      data_out : out std_logic_vector(7 downto 0);
      we : out std_logic;
      end_flag : out std_logic);
end filterkernel;

architecture Behavioral of filterkernel is
signal in1, in2, in3, in4, in5, in6, in7, in8, in9 : std_logic_vector(7 downto 0);
signal started, loaded, finish : std_logic;
begin
process (clk, rst, en, start)
variable summax, summay, summaxy1, summaxy2 : std_logic_vector(12 downto 0);
variable summa1, summa2, summa3, summa4 : std_logic_vector(12 downto 0);
variable summa : std_logic_vector(12 downto 0);
variable summanorm : unsigned(12 downto 0);
begin
if (clk'event and clk = '1') then
  if (rst = '1' or en = '0') then --restart all the procedure
    finish <= '0';
    we <= '0';
    started <= '0';
    loaded <= '0';
    data_out <= (others => '0');
    in1 <= (others => '0');
  else
    -- code
  end if;
end if;
```

168
else if (en = '1' and start = '1') then
    started <= '1';
else if (en = '1' and started = '1' and loaded = '0') then
    loaded <= '1';
in1 <= p1;
in2 <= p2;
in3 <= p3;
in4 <= p4;
in5 <= p5;
in6 <= p6;
in7 <= p7;
in8 <= p8;
in9 <= p9;
else if (en = '1' and loaded = '1') then
    summa := ("00000" & in1) + ("0000" & in2 & '0') + ("00000" & in3) + ("0000" & in4 & '0') + ("000" & in5 & "00") + ("0000" & in6 & '0') +("00000" & in7) + ("0000" & in8 & '0') + ("00000" & in9);
    summanorm := unsigned(summa)/16;
    if mask_val = '1' then
        data_out <= std_logic_vector(summanorm(7 downto 0));
    else
        data_out <= p5;
    end if;
    we <= '1';
    finish <= '1';
    started <= '0';
    loaded <= '0';
else if finish = '1' then
    finish <= '0';
    we <= '0';
end if;
end if;
end process;
end Behavioral;
B.29  Histogram Low-Pass Filter

lowpass_filter.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

-- Low pass filter of length 5 bins (not counting first and last bin)

entity hist_lowpass_filter is
  port (clk : in std_logic;
         rst : in std_logic;
         en : in std_logic;
         data_in : in std_logic_vector(18 downto 0);
         addr_in : out std_logic_vector(7 downto 0);
         addr_out : out std_logic_vector(7 downto 0);
         data_out : out std_logic_vector(18 downto 0);
         we : out std_logic_vector(0 downto 0);
         end_flag : out std_logic);
end hist_lowpass_filter;

architecture Behavioral of hist_lowpass_filter is
  type data_array is array(4 downto 0) of unsigned(18 downto 0);
  signal bins : data_array;
  signal dout : unsigned(18 downto 0);
  signal addr_in, addr_out : unsigned(7 downto 0);
  signal finish : std_logic;
  signal wea : std_logic_vector(0 downto 0);
begin
  process(clk, rst, en)
  variable counter : integer range 0 to 265;
  variable endcounter : integer range 0 to 6;
  begin
    if clk'event and clk = '1' then
      if rst = '1' or en = '0' then
        bins <= (others => (others => '0'));
        addrin <= "00000001";
        addrout <= "00000001";
        dout <= (others => '0');
        finish <= '0';
        wea <= (others => '0');
        counter := 0;
        endcounter := 0;
      elsif addrin < "11111110" then
        counter := counter + 1;
        bins(0) <= unsigned(data_in);
        bins(1) <= bins(0);
        bins(2) <= bins(1);
        bins(3) <= bins(2);
        bins(4) <= bins(3);
        addrin <= addrin + 1;
        if counter > 3 then
          wea <= (others => '1');
        end if;
      end if;
    end process;
end Behavioral;
addrout <= addrout + 1;
end if;
if counter = 3 then
dout <= (bins(0) + bins(1))/2;
elif counter = 4 then
dout <= (bins(0) + bins(1) + bins(2))/3;
elif counter = 5 then
dout <= (bins(0) + bins(1) + bins(2) + bins(3))/4;
else
dout <= (bins(0) + bins(1) + bins(2) + bins(3) + bins(4))/5;
end if;
elif addrout < addrin then
endcounter := endcounter + 1;
addrout <= addrout + 1;
if endcounter = 1 then
dout <= (bins(0) + bins(1) + bins(2) + bins(3))/4;
elif endcounter = 2 then
dout <= (bins(0) + bins(1) + bins(2))/3;
elif endcounter = 3 then
dout <= (bins(0) + bins(1))/2;
end if;
else
wea <= (others => '0');
finish <= '1';
end if;
end process;
addr_in <= std_logic_vector(addrin);
addr_out <= std_logic_vector(addrout);
data_out <= std_logic_vector(dout);
we <= wea;
end_flag <= finish;
end architecture Behavioral;
B.30 Square root operations package

customized_operations_pkg.vhd

library IEEE;
library floatfixlib;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all; -- for UNSIGNED
use floatfixlib.fixed_pkg.all;
use floatfixlib.math_utility_pkg.all;

package customized_operations_pkg is
-- Function obtained from: http://vhdlguru.blogspot.ca/2010/03/vhdl-function-for-finding-square-root.html
function sqrt(d : unsigned) return unsigned;
-- Modification of the function sqrt to work with fixed point values
function ufixed_sqrt(d : ufixed) return ufixed;
end customized_operations_pkg;

package body customized_operations_pkg is

function sqrt(d : unsigned) return unsigned is
variable a : unsigned(31 downto 0) := d; --original input.
variable q : unsigned(15 downto 0) := (others => '0'); --result.
variable left, right, r : unsigned(17 downto 0) := (others => '0'); --input to adder/sub.r-remainder.
variable i : integer := 0;

begin
for i in 0 to 15 loop
right(0) := '1';
right(1) := r(17);
right(17 downto 2) := q;
left(1 downto 0) := a(31 downto 30);
left(17 downto 2) := r(15 downto 0);
a(31 downto 2) := a(29 downto 0); --shifting by 2 bit.
if (r(17) = '1') then
  r := left + right;
else
  r := left - right;
end if;
qu(15 downto 1) := q(14 downto 0);
qu(0) := not r(17);
end loop;
return q;
end sqrt;

function ufixed_sqrt(d : ufixed) return ufixed is
--variables

variable a : ufixed(19 downto -20) := d; -- original input
variable q : ufixed(9 downto -10) := (others => '0');   -- result
variable left, right : ufixed(21 downto 0) := (others => '0');
                -- input to adder/sub. r - remainder.
variable r : ufixed(22 downto 0) := (others => '0');
variable i : integer := 0;
begin
for i in 0 to 19 loop
    right(0) := '1';
    right(1) := r(21);
    right(21 downto 2) := q(9 downto -10);
    left(1 downto 0) := a(19 downto 18);
    left(21 downto 2) := r(19 downto 0);
    a(19 downto -18) := a(17 downto -20);           -- shifting by 2 bit.
    if (r(21) = '1') then
        r := left + right;
    else
        r := left - right;
    end if;
    q(9 downto -9) := q(8 downto -10);
    q(-10) := not r(21);
end loop;
return q;
end ufixed_sqrt;
end customized_operations_pkg;