Efficient Watershed Algorithms for Image Segmentation and Related Prototype Architectures

A

Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

By

C. RAMBABU

Department of Electronics and Communication Engineering
Indian Institute of Technology Guwahati
Guwahati - 781 039, INDIA.

April, 2004

THIS WORK IS DEDICATED TO

MY PARENTS

AND

WIFE

Indian Institute of Technology Guwahati
North Guwahati, Guwahati -- 781 039
Assam. INDIA.
Ph: (+91) 361-2582504 (Office)
Fax: (+91) 361-690762
Email: indra@iitg.ernet.in, indrac@postmark.net

Dr. Indrajit Chakrabarti
Assistant Professor
Department of Electronics & Communication Engineering.

Certificate

This is to certify that the thesis entitled ``**Efficient Watershed Algorithms for Image Segmentation and Related Prototype Architectures**", submitted by C. Rambabu, a research student in the *Department of Electronics and Communication Engineering, Indian Institute of Technology, Guwahati*, for the award of the degree of **Doctor of Philosophy**, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the Institute and in my opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Indrajit Chakrabarti

(Indrajit Chakrabarti)

22/4/2004

Acknowledgements

I feel it is a great privilege to express my deepest and most sincere gratitude to my supervisor, Dr. Indrajit Chakrabarti for his suggestions, constant encouragement and support during the course of the thesis work. I am also grateful to the other members of my doctoral committee, namely Prof. Anil Mahanta, Dr. P.K. Bora and Dr. J. S. Sahambi for their valuable comments on my work. I thank Dr. D. Ghosh for his comments on my thesis work. I take this opportunity to thank Prof. A. K. Gogoi, the head of the department for his kindness in allowing me to use various computing facilities of the department. I would also like to thank the other faculty members of the department for their encouragement and help. I am thankful to Prof. S. Nandi, Dept. of Computer Science & Engineering for introducing me to VHDL and various hardware packages.

I thank Dr. P. K. Bora, the Associate Dean of Academic Affairs, IIT-Guwahati for sanctioning partial financial support for my travel to Singapore to attend ICICS-PCM 2003 conference. I also thank Prof. M. K. Choudhuri, the Dean of Research & Development, IIT-Guwahati, CSIR and IEEE Hong Kong Chapter for supporting my visit to two conferences, namely ICICS-PCM'03, Singapore and ICFPT'02, Hong Kong. I express my thanks to Dr. D. Ghosh, for accommodating me and for his hospitality during my stay in Singapore.

I am also grateful to all the members of the non-teaching staff of the department, without whose help I could not have completed this assignment. I thank all my fellow research students and M. Tech students for their cooperation. My thanks also go out to all my friends, namely P. Vinod, C. Prem Kumar, Nirmal Kumar, N. Sathyanarayana, V. Naveen Kumar and I. Sunil who have made my stay at Guwahati a memorable period of my life.

Finally, I must express my heart-felt gratitude to my wife Lalitha Prasanna Kumari for her constant encouragement and support throughout my research work and to my parents for their countless sacrifices, unbounded love and support all through.

C. Rambabu 22/04/2004

(C. Rambabu)

Abstract

Image segmentation is concerned with decomposing a given image into its constituent regions or objects. It is an important preliminary step in diverse fields of application like object recognition, image compression, medical image processing and biological analysis methods. Among the existing image segmentation algorithms, watershed transformation is deemed to be a powerful tool for image segmentation owing to the simplicity of its formulation and implementation and its ability to identify the important closed contours of a given image. However, the time complexity of the majority of the watershed transform algorithms is quite high, making their real-time application difficult. In particular, real-time processes like moving object segmentation, road traffic monitoring and analysis of steel fracture demand fast computation of watershed transformation for image segmentation. At the same time, a dedicated hardware architecture for implementing watershed algorithms would give rise to faster results as compared to a software program executed on a general purpose processor.

The work embodied in the present thesis is concerned with formulating improved versions of a few standard watershed algorithms and developing appropriate prototype hardware architectures for the same. Following results have emerged in the process.

- An improved algorithm has evolved from Meyer's flooding-based watershed algorithm based on ordered queues. Hardware implementation of Meyer's algorithm, which is based on 256 ordered queues, is very complex and expensive. The proposed algorithm, however, uses a single queue thereby reducing the computational complexity but at the same time showing almost identical performance compared to Meyer's algorithm. Moreover, a prototype architecture has been developed for effective realization of the proposed method.

- A drawback of the above proposed algorithm is that no synchronization is possible in a non-minimum plateau having more than one closely located regional minima. Moga's technique, based on lower complete transformation, can be applied to tide over this problem. However, this would entail time-consuming multiplication by a constant of a gradient image based on geodesic distance. The present work evolves a hillclimbing-based method which improves upon Moga's algorithm by avoiding the computationally expensive lower complete transformation. Moreover, the regional minima detection and labelling is performed in one scan rather than two scans by the use of two queues. The reduced complexity makes the proposed algorithm amenable for hardware

realization. An FPGA-based architecture has been developed to implement the algorithm with moderate hardware complexity.

- The work reported in the present thesis has gone on to develop an efficient immersion-based algorithm based on Vincent and Soillé. Undesired effects like thick watershed lines and isolated minima are not uncommon during the process of flooding of catchment basins and detection of watershed pixels. In the proposed algorithm, flooding starts from pre-determined regional minima, during which the label of the neighbouring pixels are decided on the basis of conditional neighbourhood comparisons and geodesic distance from the outer brim of the nearest plateau. As a result, always continuous watershed lines with a uniform width of a single pixel are produced. A prototype hardware architecture for this algorithm too has been developed.

Contents

List of Figures

List of Tables

Nomenclature

1-D	One-dimensional
2-D	Two-dimensional
3-D	Three-dimensional
CA	Cellular Automata
FIFO	First In First Out
FPGA	Field Programmable Gate Array
INIT	Initialization label
IP	Immersion Process
LMD	Local Minima Detection
LP	Label Propagation
MAX_DIST	Maximum allowed lower distance value
MIMD	Multiple Instruction Multiple Data
MINIMUM	flag for a one pixel wide regional minimum
MR	Magnetic Resonance
MRF	Markov Random Field
NARM	Not A Regional Minimum
NON_MINIMUM	flag for non-minimum pixel

NOT_VISITED Flag for pixels which have not been reached by the current computation

ON_PLATEAU flag for pixels on the plateau surrounded by no lower neighbors

OQ Ordered Queue

SIMD Single Instruction Multiple Data

SKIZ Skeleton by Influence Zones

SPMD Single Program Multiple Data

VISITED Flag of pixels already visited by the current computation

WSHED Watershed label

Mathematical Notations

λ	Wavelength
$\delta_B(f)$	Dilation of a function f by a structuring element B.
$\varepsilon_B(f)$	Erosion of an image f by a structuring element B.
$\gamma_g(f)$	Opening of an image f by a structuring element g.
$\phi_g(f)$	Closing of an image f by a structuring element g.
$\delta_f^{(1)}(g)$	Geodesic dilation of f from g.
$\varepsilon_f^{(1)}(g)$	Geodesic erosion of f from g.
$\gamma_f^{(rec)}(g)$	Dilation-based reconstruction of f from g.
$\phi_f^{(rec)}(g)$	Erosion-based reconstruction of f from g.
$C_h(F)$	Set of the catchment basins of image F at level h.
D_F	Domain of image F.
$D_H(S_1 \Rightarrow S_2)$	Directional hamming distance from S_1 to S_2.
d	Lower distance image.
$d(p)$	Lower distance value of pixel p.
$dist$	Distance image.
F	Gradient image.
$flags$	Flag image.

G	Grid
$G_e(f)$	Gradient by erosion of f.
$G_d(f)$	Gradient by dilation of f.
$G(f)$	Morphological gradient of f.
H	The heighest existing graylevel in an image.
h	graylevel in the image.
$IZ_A(B)$	Influence zones of set B inside A
L	Label image
$L(p)$	Label value of pixel p.
$l(p)$	Lower-complete transform of pixel p.
$MG(f)$	Multi-scale gradient of an image f.
MIN_h	The regional minima at an altitude h.
$\mathbb{N}$	Set of integer numbers.
N_G	Neighborhood pixels on the grid G.
$N_G(p)$	Set of neighboring pixels of p on the grid G.
$N_G^-(p)$	Past neighborhood of p.
$N_G^8(p)$	8-connected neighbors of the center pixel C on the grid G.
n	The number of image pixels.
$Outers$	Array of outer pixels.
P	Performance measure.
P_M	Number of minima plateaus in an image.
P_N	Number of non-minima plateaus in an image.

$SKIZ_A(B)$	Skeleton by influence zones of B inside A.
sln	Steepest lower neighbor image.
$sln(p)$	Steepest lower neighbor value of pixel p.
slc	Steepest lower complete image.
$slc(p)$	Steepest lower complete value of pixel p.
$T_h(F)$	The threshold of image F at gray level h.
$U[f]$	Umbra transform of f.
$Wshed(F)$	Watershed of image F.

Chapter 1

Introduction

Image segmentation is an important step in most image analysis tasks such as object recognition, object-based image compression and context-based indexing of images. The general segmentation problem involves the partitioning of a given image into a number of homogeneous segments (spatially connected graphs of pixels). Alternatively, segmentation can be considered as a pixel labelling process in the sense that all pixels that belong to the same homogeneous region are assigned the same label. The segmentation process can be based on finding the maximum homogeneity in grey levels within the regions identified. The mathematical definition of image segmentation based on region is given by

$$\begin{aligned} \bigcup_{i=1}^{n} R_i &= R, \quad R_i \text{ is a connected region} \\ R_i \cap R_j &= \phi, \quad \text{for all } i \text{ and } j,\ i \neq j, \text{ and } \phi : \text{null set} \\ R_i &: \text{a connected region, } i = 1, 2, \ldots, n. \end{aligned}$$

where R : the region encompassing the entire image, consisting of the union of n disjoint regions.

Segmentation of complex images is a difficult problem. Success of an automated image analysis procedure depends to a large extent on the accuracy of the segmentation technique being followed. This introductory chapter begins with a section giving a brief survey of different image segmentation algorithms. Section 1.2 provides an outline of the watershed transform approach to image segmentation. Motivation of the present work is given in the subsequent section. The chapter ends with a brief outline of the present dissertation.

1.1 Algorithms for Image Segmentation

In place of a single standard method, a collection of techniques have been put forward as possible solutions to the image segmentation problem. The segmentation algorithms are usually based on discontinuity or similarity property of intensity values. Although it is a very difficult problem, several algorithms [HS85, PP93, Pra03b] have been proposed for its solution. They can be broadly grouped as histogram-based algorithms [Wes78, Ots78, MH88, SAB99], edge-based algorithms [ARH80, Pra80, Per80, Can86], region-based algorithms [BG89, Wu93, BM93, CL94, Mog97, RM01], Markov Random Field-based algorithms [DE87, SS89, DJ89, DJNC90] and clustering-based methods [Kur91, OM97, PF99].

1.1.1 Histogram-based Algorithms

These algorithms make decisions based on local pixel information, and are effective when the intensity levels of the objects are outside the range of intensity levels in the background. The idea is to make a histogram of the pixel values of the image from which an appropriate point for thresholding is found. The thresholding operation is regarded as the partitioning of pixels of an image into two classes: the object and the background. It is based on the assumption that the image has a bi-modal histogram and, therefore, contains an object or objects of interest that can be extracted from the background by a simple operation that compares image values with a threshold. Thresholding techniques can be classified as bi-level thresholding and multi-level thresholding. In bi-level thresholding, the image is subdivided into two regions, namely object (black) and background (white). This technique is very effective in the case of the images having a uniform brightness against the background of relatively different brightness. Typical examples include handwritten and typewritten text, microscopic and biomedical images. The luminance is a distinguishing feature that can be utilized to separate the object from the background. When the image is composed of several objects, one needs several thresholds for segmentation. This is known as multi-level thresholding. If the image is composed of regions with different grey level ranges, the histogram of the image usually shows different peaks, each peak corresponding to one region and the adjacent peaks are separated by a valley. The bottom of the valley is the threshold that can be used to separate objects belonging to one region from those of the other.

Several analytic approaches to the setting of a luminance threshold have been proposed [Wes78, Ots78, SAB99]. A recursive approach for image segmentation has been proposed by Cheriet *et al* [CSS98]. This approach selects the brightest homogeneous object from a given image at each recursion,

by leaving out the darkest homogeneous object. At each iteration, the histogram of the image is drawn and the largest peak is separated from the rest of the image. The process is continued until there are no more peaks in the histogram. This method performs well on images with two classes. Weska *et al* [WNR74] have suggested the use of a Laplacian operator in selection of the luminance threshold. A grey value histogram formed of only those pixels of the original image, which lie at the coordinates corresponding to very high or very low values of the Laplacian, tends to be bi-modal with a distinct valley between the adjacent peaks. Banu and Faugeras [BF82] have proposed a gradient relaxation method for solving the histogram unimodal problem. This approach is based on assigning to each pixel an initial probability of occurrence and then using the gradient relaxation based on compatibility function that takes into account the relationship between a pixel and its eight immediate neighbors. The relaxation process is iterative where pixel values are changed so that the histogram is no longer unimodal and the threshold can be easily detected. This technique provides a better control by defining the parameters that are under user control.

1.1.2 Edge-based Algorithms

These methods use the output of an edge detector to derive the meaningful regions. It is possible to segment an image into regions of common attributes by detecting each region for which there is a significant change in the attributes across the boundary. The resulting boundary appears to be visually correct when overlaid on the original image. A large variety of methods are available in literature [GW01, Pra03a] for edge detection. If an image is noisy or if the region attributes differ by a small amount between regions, a detected boundary may often be broken. Edge linking methods can be used to bridge small gaps in such a region boundary.

Prager [Pra80] has proposed a set of algorithms used to perform segmentation of natural scenes through boundary analysis. The goal of his algorithm is to locate the boundaries of an object correctly in a scene. Perkins [Per80] uses an edge-based technique for image segmentation. The edge-based segmentation has not been very successful because of small gaps that allow merging of dissimilar regions. In order to alleviate these problems, Perkins's method proposes an expansion-contraction technique in which the edge regions are expanded to close the gaps and then contracted after the separate regions have been labelled.

1.1.3 Region-based Algorithms

Region growing is one of the conceptually simplest approaches to image segmentation. A range of image segmentation algorithms are based on region growing. These algorithms take one or more pixels, called seeds, and grow the regions around them based upon a some homogeneity criterion. If the adjoining pixels are similar to the seed, they can be merged within a single region. The process continues until all the pixels in the image have been assigned to one or more regions.

Chang and Li [CL94] have proposed a region-growing technique for image segmentation. In this method, an image is first divided into many small primitive regions which are assumed to be homogeneous and then merged to form larger regions until no more merging operations are possible. This process is guided by an analysis of local features. Moreover, this algorithm is robust and produces high-quality segmentation on a wide range of textured and grey scale images. Alternatively, a hierarchical stepwise optimization algorithm for region merging has been proposed by Beaulieu and Goldberg [BG89]. This algorithm is based on stepwise optimization and produces a hierarchical decomposition of the image. The algorithm starts with an initial image partition into a number of regions. At each iteration, two segments are merged provided they satisfy the criterion of merging a segment with another. The advantage of this algorithm over the non-hierarchical methods is that there is no need to specify the seed points.

The watershed transform technique also belongs to the broad class of region-based segmentation approach. Various watershed algorithms based on the topographical concept exist in the literature [BM93, Mog97, Vie96, RM01, NC03]. A detailed survey of segmentation techniques based on watershed transform is given in section 1.2.

1.1.4 Markov Random Field-based Algorithms

Recently, the Markov Random Field (MRF) based segmentation techniques [DJNC90] have become popular because MRF provides a powerful means of characterizing different region classes. From the computational perspective, the local property of MRF leads to algorithms which can be implemented in a local and massively parallel manner. For these reasons, MRF-based methods have been widely applied for image modelling, compression and classification. However, in the absence of *a priori* information such as the number of regions and its parameters, these methods are difficult. The main difficulty is that the model and its parameters are unknown and need to be computed from the given image before segmentation. To compute the parameters effectively, the segmented image itself is needed.

A possible solution to this problem is iterative segmentation and estimation. One first estimates the parameters in small regions to obtain a crude segmentation. Next, one estimates the parameters again from this segmented image. This is repeated until the termination criterion is satisfied. Although these methods have all demonstrated some degree of success, in practice, the performance of such methods may be hampered by the number of parameters. For example, in case of segmentation of a noisy and blurred image, parameters such as mean and variance of noise and blurring matrices need to be defined for each region. Several variations of MRF-based segmentation algorithms have been proposed in literature [DE87, DE87, DJ89, SS89]. They differ in the following parameters, namely, the type of the prior model, the model for corrupting the true image by noise, the type of the optimization algorithm, and the details of the algorithms themselves.

1.1.5 Clustering-based Algorithms

Image segmentation can be performed effectively by clustering the image pixels. The idea of segmentation by clustering is simple. However, it is computationally expensive. Cluster analysis allows the partitioning of data into meaningful sub-groups and it can be applied for image segmentation. Clustering analysis either requires the user to provide the initial seeds for the regions to be segmented or uses non-parametric methods for finding the salient regions without the need for seed points.

An efficient clustering algorithm has been proposed by Kurita [Kur91]. The algorithm starts with an initial partition of a given image into a number N of segments and sequentially reduces the number of segments by merging the best pairs of segments among all possible pairs in terms of a given criterion. Pauwels *et al* [PF99] have proposed a non-parametric clustering algorithm for image segmentation. It is able to handle unbalanced and highly irregular clusters using intermediate level processing. First, a density image is formed by convolving the image data with a Gaussian density kernel and then the candidate clusters are identified by using the gradient ascent and each point is linked to the point of the highest density among its neighbors. The authors have concluded that the algorithms would produce a smooth version of equalization algorithm without destroying any information in the image. Ohm and Ma [OM97] have proposed a feature-based cluster segmentation method for image sequences. This algorithm analyzes specific features from the image sequence and checks their reliability and evidence locally for images in order to build segments that are probably part of the object. The segmentation procedure is basically a clustering procedure which takes into account different features such as color and motion. The pixel feature vector is then compared to a set of cluster feature vectors and hence

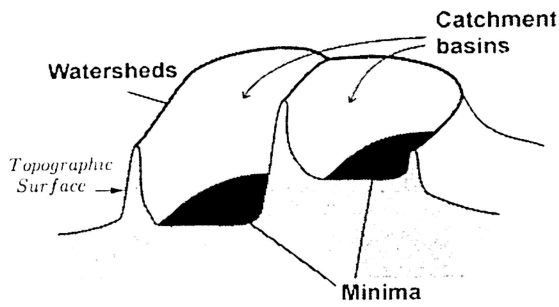


Figure 1.1: Watersheds, catchment basins and minima

classified into a feature class to generate clusters. The set of cluster feature vectors is updated for each image in the sequence which is also used for the segmentation of the next image. The authors have further concluded that the algorithm is a hybrid combination of a block-recursive technique and a pixel-recursive technique and produces a dense vector field.

1.2 Watershed Transform Approach

Among the existing segmentation algorithms, watershed transformation has proved to be a very useful and powerful tool for morphological image segmentation because of its straightforward formulation and implementation as well as its ability to identify the important closed contours of a given image. The idea of the watershed transformation is quite simple. As depicted in Figure 1.1, a gradient image obtained by applying an appropriate gradient operator on a grey scale image is considered to be a topographic surface, where the grey scale value of a pixel denotes the altitude of that pixel. Let a drop of water fall on the topographic surface. By gravity, it would move to a lower altitude until it reaches a local altitude minimum (regional minimum). The whole set of points of the surface, whose steepest path reaches a minimum, constitute the *catchment basin* [BL79, Beu90a] associated with this minimum. All the points that drain into a common catchment basin are part of the same catchment basin. Each pixel in this digital image is assigned a label during the transformation of the catchment basin of a regional minimum. The watersheds are the ridges dividing the adjacent catchment basins.

Two intuitive approaches exist for computing the watershed transformation. The first one, known as rainfalling, starts finding the local minima throughout the image. Each local minimum is given a unique label. Next, a water drop is placed at each non-labelled pixel. The drop moves to its neighbor with lower amplitude until it reaches a labelled pixel. A catchment basin is then defined as the set of pixels whose respective downstream paths all end up in the same labelled local minimum. Although

the definition stated above is quite intuitive, it is not very suitable for practical implementation.

In an alternative approach known as flooding, conceptual single pixel holes are pierced at each local minimum, and the topograghic surface is immersed into a large body of water. The catchment basins will fill up with water, starting at these local minima, and at points, where water arising from different basins would meet, dams are built. On non-minima flat regions, wavefronts coming from distinct sources advance synchronously, one step at a time. When the water level has reached the highest peak in the landscape, the process is stopped. As a result, the landscape is partitioned into regions of basins separated by dams. The dams are the watershed lines or simply watersheds. The watershed transformation thus denotes labelling of an image, such that all points of a particular catchment basin have the same unique label, and a spatial label, distinct from all labels of the catchment basins, is assigned to all points of the watershed. Various definitions for watersheds in both continuous and digital space have been proposed in the literature [Beu90a, VS91, BM93, RM01, NC03].

If the input image has been transformed into a form in which the minima mark the relevant objects and the crest lines correspond to the image boundaries, the watershed transform will partition the image into meaningful regions. This achieves a useful result for image segmentation. The watershed of a given gradient image has several useful properties as follows:

- The watershed lines form closed and connected regions.
- There is no intersection between the regions.
- The union of the segmented regions and the watershed lines separating them, constitutes the whole surface.
- Each region contains a single regional minimum separated as a single pixel.

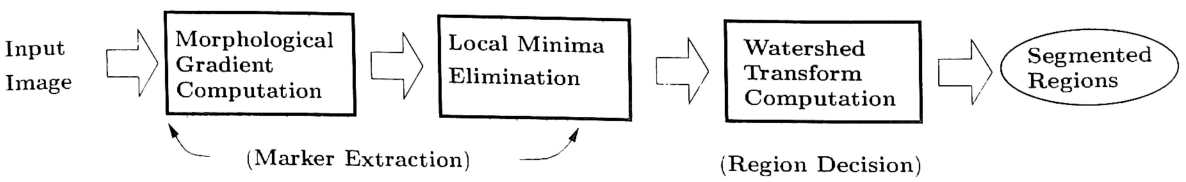


Figure 1.2: Flow diagram of typical watershed-based image segmentation

Figure 1.2 shows the procedures for applying watersheds to region analysis. It is known that the watershed transform of the gradient usually produces over-segmentation (many small catchment basins

due to additive noise in natural scene images. The performance of the watershed transformation depends largely on the algorithm used to compute the gradient. The conventional gradient operator like those due to Sobel [Dav75], Prewitt [GW01] and Canny [Can86] generally produce small gradient modulus values, which may originate from additive noise or quantization error. To alleviate this problem, Wang [Wan97] has proposed a multi-scale morphological gradient operator, which effectively enhances the blurred edges and reduces the number of local minima. A multi-scale gradient image [Wan97, CKP98] is used as an input to the watershed transform algorithms in the present work. The number of segmented regions can be controlled by using a multi-scale gradient algorithm, as discussed in Chapter 2.

1.2.1 Applications of Watershed Transform Techniques

Watershed transformation has been proved to be a very useful and powerful tool in many different fields. Although the initial application of watershed algorithms was in the field of topography, they have been more popular in different industries, biomedical signal processing [ADR95], medical image processing [WHOF96, RDKJ02] and computer vision [Gee95]. As examples, one can mention here the contribution of watersheds to an automatic system of analysis of images acquired during oil exploration, road traffic analysis and monitoring [BBY92, BM93], to the extraction of cleavage facets from a fractured surface in steel and to segmentation of several types of images including overlapping grains and 3-D holographic images. Watersheds have also been used in extracting and tracking objects in time sequence of a moving heart in nuclear medicine, in counting objects in noisy, anisotropic 3-D biological images [ADR95], and for segmenting the internal structure in 3-D magnetic resonance (MR) images of brain [WMTM95]. In addition, various results on exploiting the watershed algorithm for segmentation of color and multi-spectral images, for object-based video coding [Wan98, Gro97, SYL03], and for remote sensing systems of satellite and radar images have been reported. Many other examples of segmentation based on watershed transform can be found in the literature [Beu90a, MB90, Beu90b, BM93].

1.2.2 Review of the Different Classes of Watershed Algorithms

Watershed transform has been approached from two different perspectives. The first group of algorithms aim to detect the catchment basin by simulating the flooding process [BL79, MB90, BM93, Mey94, VS91, RM01] while the second group directly track the watershed lines by using arrowing techniques [Beu90a, BM93]. More efforts have been spent on the first group of algorithms, however.

Among these, there are algorithms which construct the watershed lines and those which do not delimit the catchment basins by watersheds (0-width watershed lines). The first group of algorithms can be subdivided into two classes, namely the serial or ordered watershed algorithms and parallelized watershed algorithms.

Serial Watershed Algorithms

In a sequential algorithm, the newly computed label of a pixel will serve as the argument for the transformation of its not yet transformed neighbors. The result of the transformation depends completely on the scanning order. Generally, for simplifying the access to the image memory, one adopts forward and backward raster scanning. The ordered algorithms are essentially the same as the sequential algorithms except for the scanning order of the pixels.

It was Beucher and Lantuéjoul [BL79] who proposed an immersion-based watershed algorithm. A non-parametric technique has been developed for contour extraction in a grey scale image. This method relies on defining the contours as the watersheds of the gradient modulus of the illumination function (considered as a topographic relief surface). Among the broad classes of sequential algorithms for computing the watershed transforms, a fast implementation of an immersion-based watershed algorithm was proposed by Vincent and Soillé [VS91]. The authors simulate a flooding process, in which water comes up out of the ground flooding the catchment basins, without determining the regional minima beforehand. This method examines all the pixels in an image at successive grey scale values, and operates in two stages, namely a sorting step followed by a flooding step. Utilizing a First-In-First-Out (FIFO) data structure, the pixels at altitude $h+1$ are processed after processing those at altitude h, starting from the lowest altitude. At each grey level, pixels appended to the catchment basins are located in their immediate neighborhood at lower grey levels. All pixels at the current grey level, that have at least one neighbor at a lower level, are put in a FIFO queue and assigned the label of the neighbor(s) with the lowest grey level. This divides the problem into n subproblems, where n is the number of all present pixel altitudes. Due to the processing of pixels at an altitude h in every iteration, the problem is reduced to calculating the geodesic *skeleton by influence zones* (SKIZ). Whenever two floods originating from different catchment basins reach each other, a dam (watershed) is built to prevent the catchment basins from merging. This algorithm always produces incomplete watershed lines and isolated minima.

Alternative sequential watershed algorithms based on an ordered queue have been proposed in [BM93, Vie96]. These algorithms are based on a definition of flooding by watersheds and a two-level

ordering relation is provided by an ordered queue (OQ) which in fact, embeds 256 queues corresponding to all possible gray levels of any pixel. The ordering relation is based on listing the pixels in the order in which they will be flooded. Each pixel is put into the queue corresponding to its grey level. The OQ is initialized so that at least one point in the queue belongs to a regional minimum. Starting from these regional minima, recursive label propagation is performed using an ordered queue. It yields a complete tessellation of the image into its homogeneous regions without producing any watershed lines.

Finally, several shortest-path algorithms for the watershed transformation with respect to topographical distance can be found in literature [MB90, Mey94]. In fact, these methods compute the shortest paths between the regional minima and all other pixels. Here, a new definition of watershed and catchment basins in terms of a distance function and topographic distance has been introduced [Mey94, Vie96]. Thus, a pixel is incorporated in the catchment basin associated with the "closest" regional minimum. Consequently, each pixel has an ancestor according to the flooding order, which is a neighbor with the precedent distance value. These shortest paths are tracked by walking upward (hill-climbing) or downward (rainfalling) on the topographic surface according to the precedence relation imposed by distance. In the rainfalling simulation, a non-minimum pixel is labelled on a path along which a drop of water would slide towards a minimum, when it starts falling from that non-minimum point in the topographic surface.

On the contrary, in the hillclimbing simulation, regions grow independently around their regional minima. The labels of minima are propagated upward to the higher neighboring pixels which do not have neighbors along a steepest path other than the given candidate for region growing. A hillclimbing based watershed algorithm has been introduced by Meyer [Mey94]. In this method, the regional minima are first computed and then the adjacent pixels of these minima are added to a hierarchical queue. At each iteration, the pixels with the lowest altitude are dequeued from the queue and then one labels all the upper neighbors of the current pixel along the steepest slope, unless the neighboring pixel is already labelled and the label differs from the current pixel label, in which case the neighboring pixel is classified as a watershed pixel. This step is repeated until all the pixels are processed, thus simulating an over-flooding of the processed data (called *hillclimbing*). These algorithms provide a straightforward way to find the watershed lines but require extensive computations. In general, the immersion-based methods of the type discussed earlier are much faster than the shortest path-based algorithms.

Parallelized Watershed Algorithms

Parallel implementation of the watershed transform can be classified as domain decomposition and functional decomposition. In the domain decomposition, one distributes the image over the processors in a regular way (static mapping) and uses a sequential algorithm for the subimage in each subdomain. Next, one inserts the synchronous and communication points where the results would depend on the neighboring subdomains. The subresults are then merged to obtain the final solution. On the contrary, in a functional domain, one distributes the local minima among the processors while simulating flooding from local minima. In this case, the efficiency depends on the number of local minima, and the size of the corresponding catchment basins.

A standard classification of parallel computer systems consists of four classes. The two categories often encountered are SIMD (single instruction, multiple data) and MIMD (multiple instruction, multiple data). In a SIMD computer, all processor elements simultaneously execute the same operation on different data items, whereas in a MIMD machine the processors may execute different operations on their own data. Both SIMD and MIMD computers can possess either the shared memory or the distributed memory. Various parallel programming models exist. In a message passing environment, tasks are created, which interact by sending and receiving messages. The approach most often used is called SPMD (single program multiple data), which implies that every processor runs the same program, performing operations on its own data space. In a shared-memory programming model, the tasks share a common address space.

In the case of watershed algorithms, usually domain decomposition is used on distributed memory architecture. Granularity depends on the distribution of data among the processors, the number of processors and the image content. Each processor executes the same code on the subimage it encapsulates. This approach is called SPMD (single program multiple data). Each SPMD implementation of watershed splits the computation into two parts. First, partial results are produced by local flooding in each processor with any of the existing techniques. Then, the subresults are merged into the final value. Each processor has access to an overlap region between the subdomains, determined by the neighborhood of each boundary pixel. Pixels in the boundary regions are manipulated only by the process to which the subdomain is assigned, but are available for reading by the processors of neighboring subdomains. Parallelization of watershed algorithms which simulate flooding and do not construct watershed lines are introduced in [Mog97].

Parallelization of the watershed transform by ordered queue has been discussed in [Mog97]. The

program uses the SPMD approach with synchronization by messages from or to a master process. The image is distributed in blocks. In the step concerning the local minima, plateaus are examined by breadth-first scans in each subimage using a FIFO queue. If a plateau is spread over different subdomains, communication between processors is necessary during which merging of the parts takes place. This may require repeated communication until stabilization. During the flooding process, each processor performs flooding in its own subdomain based on an ordered queue, as in the sequential algorithm. To allow flooding to propagate to neighboring subdomains, one first performs local flooding at all grey levels in the subdomain, followed by communication and re-flooding until the label propagation stabilizes.

An efficient watershed segmentation algorithm suitable for parallel implementation has been proposed by Moga *et al* [MCG95,Mog97]. Local minima detection with lower completion on non-minima plateaus requires communication until stabilization. The flooding step is considered as labelling of each point in the lower complete graph by assigning to it the label of the local minimum which is located on the steepest path from the pixel. Each non-minimum pixel gets a label by "walking downward" on a steepest slope line towards a minimum. A steepest slope line is either determined by a single process, if the whole path belongs to the workspace of the same process, or remains unresolved if the path extends over several subdomains. Paths are first locally traced in each process and resolved if possible. Then, in the *linkage* step, the unresolved subpaths are pasted with the resolved ones from the adjacent subdomains. Neighboring processes exchange labels of the pixels within the edges to allow the paths to extend in the adjacent subdomains, if needed. A process keeps labelling, communicating and completing the paths as long as there are unresolved paths. When all the processes become active, all the pixels are considered to have been labelled and the algorithm stops.

Alternatively, parallelization of a hillclimbing algorithm combined with connected component labelling has been considered by Bieniek [BBM97] and by Moga *et al* [MG97]. The main idea is to solve the watershed problem independently on all subdomains without synchronization. Instead, temporary labels are assigned to the pixels which will be flooded from the adjacent subdomains. Finally, a watershed algorithm based on sequential scannings has been introduced by Moga *et al* [MVG95, MG97]. The implementation consists of repeated raster scans and reverse raster scans within the subdomains and message passing among the processors until stabilization. Labels in the boundaries between subimages have to be communicated between the processors. Even if the computation has stabilized within a subdomain, new raster scans may be necessary due to changes in the boundary region caused by other processors. A master process detects when globalization has become necessary. Although the sequen-

tial algorithm is extremely slow, the method has good scaling properties in a parallel implementation.

1.3 Motivation of the Present Work

Various watershed algorithms exist in literature [BM93, Mey94, VS91, RM01], as mentioned in th previous section. These algorithms has been proposed to be implemented on standard sequential processors. Each algorithm involves a trade-off between the speed and the data structure complexity. Unfortunately, the watershed computation is a considerably time-consuming task and therefore cannot suit real-time operations in most cases. For example, real-time applications like moving object segmentation, traffic analysis and monitoring, and steel fissure analysis call for fast image segmentation procedures. Moga has proposed parallel watershed algorithms [MCG95, MVG95, MG97, Mog97] and their implementation on coarse grain architecture (shared memory computer Cray Research T3D and Parsytec SuperCluster). In these algorithms, the image is split into equally sized blocks, each block being assigned to one processor. This implementation significantly improves the processing time when data are uniformly distributed, which is, however, not the case in general. Moreover the implementation of the architecture proposed by Moga is expensive, involving no less than 128 DEC Alpha 21064 processors in a Cray Research T3D parallel computer. This necessitates developing a dedicated hardware architecture that would in general produce faster results as compared to the software program executed on a general purpose processor.

The major objectives of the present work has been to formulate improved versions of a few important standard watershed algorithms, and develop suitable hardware prototype architectures for the same. Such implementations, if carried out in an FPGA environment, would ensure the desired speed of watershed transform computation without incurring large expense on hardware.

The major contributions of the work reported in this thesis are as follows :

- An improved watershed algorithm has been derived from Meyer's simulated flooding based algorithm by ordered queues and a prototype architecture has been developed for its effective implementation. In Meyer's watershed algorithm [BM93], simulation of the flooding process involves 256 queues which makes the hardware realization very complex and costly. The proposed scheme, however, uses only one queue and decides the labels of the neighboring pixels based on conditional comparisons. It demonstrates a reduced computational complexity but almost an identical performance when compared to Meyer's algorithm.

- However, the drawback of the above proposed algorithm is that no synchronization in label propagation is possible in a non-minimum plateau which has more than one closely located regional minimum. It leads to a consequent reduction in the performance measure. To alleviate this problem, one can follow a technique based on lower complete transformation by Moga [Mog97], which works well with regular gray-scale images. However, this would require a time-consuming multiplication by a constant of a gradient image based on geodesic distance. This transformation involves $O(m \times n)$ multiplications and additions, where $m \times n$ is the size of the input image. The present work reports a new hillclimbing-based technique which improves upon Moga's algorithm [Mog97] by doing away with the lower complete transformation. This algorithm attempts to reduce the computational complexity in two ways. First, the lower complete image computation is avoided. Secondly, the regional minima detection and labelling is computed in one scan rather than two scans by the use of two queues. The reduced complexity makes the algorithm suitable for dedicated hardware implementation. The flooding of the non-minima plateau pixels is achieved by directly using the lower distance and the steepest lower neighbor image values. Moreover, an FPGA-based architecture has been developed to implement the proposed algorithm involving a moderate hardware complexity.

- The present work also proposes an efficient immersion-based watershed computation method based on the algorithm due to Vincent and Soillé [VS91] and its prototype architecture. While trying to flood catchment basins and detect watershed pixels, several side effects such as thick watershed lines and isolated minima can occur. Though measures to tide over this difficulty in the conventional algorithm [VS91] have been suggested, they introduce non-uniformity in the region growing process. In the proposed algorithm, flooding of the catchment basins starts from the pre-determined regional minima, during which the labels of the neighboring pixels are decided based on conditional neighborhood comparisons and the geodesic distance from the outer brim of the nearest plateau. As a result, always continuous and thin watershed lines can be found quickly. The proposed algorithm shows better computational complexity and throughput than the conventional algorithm as it is able to decide labels of eight pixels at a time in contrast to Vincent's algorithm [VS91], which does so for a single pixel. Moreover, the proposed algorithm produces watershed lines possessing a uniform width of a single pixel in contrast to the algorithm [VS91] that may give rise to thick watershed lines. The proposed algorithm also derives its strength from the parallelization of the label propagation process involved in the flooding pro-

cedure. The present work also proposes a prototype architecture developed to implement the proposed algorithm.

1.4 Outline of the Dissertation

The thesis records a detailed account of evolution of improved versions of a few standard watershed based image segmentation algorithms and prototype hardware architectures for realizing the same. The organization of this thesis is given below.

Chapter 2: This chapter is devoted to the multi-scale morphological gradient operations for watershed transformation. A review of some morphological gradient operators used in image segmentation is also presented.

Chapter 3: This chapter is concerned with a proposed novel flooding-based watershed algorithm as well as its complexity analysis. The simulation results, and a quantitative comparison between the present work and the conventional Meyer's algorithm [BM93] is given. In addition, a prototype FPGA-based architecture, which is developed to implement the modified algorithm, and its implementation results are described.

Chapter 4: This chapter presents a new hillclimbing-based technique for image segmentation and its architecture. The simulation results along with quantitative comparison between the proposed algorithm and the existing algorithm due to Moga [Mog97] are presented. A prototype FPGA-based architecture, which is developed to implement the improved hillclimbing technique and its implementation results also find place in this chapter.

Chapter 5: This chapter presents a detailed discussion including analysis of complexity of a proposed improved immersion-based watershed computation method. Some of the results produced by applying the proposed and the conventional method due to Vincent and Soillé [VS91] on different standard images are given. A 2-D cellular automata (CA) based prototype architecture, which is developed to implement the improved immersion watershed algorithm, and its FPGA implementation results is also presented in this chapter.

Chapter 6: The last chapter draws conclusions from the research work reported in this thesis and suggests a few areas for further extension.

Appendix A: It provides the VHDL description of the 3 × 3 2-D CA model for conditional neighborhood comparison process involved in implementing the proposed improved immersion-based discussed in chapter 5.

Appendix B: It provides a brief description of the Xilinx FPGA device used for implementing the hardware designs pertaining to the algorithms proposed in this thesis.

Appendix C: It shows how some of the component modules of the prototype hardware architectures described in Chapters 3, 4, and 5 can be implemented by the internal logic blocks of a Xilinx FPGA device.

Chapter 2

Morphological Gradient Operations

The goal of this chapter is to describe the morphological operators generally used for image processing and the existing algorithms to find the morphological gradient image. In the context of the present work, the multi-scale morphological gradient operator is important due to its role in preventing over-segmentation while constructing the gradient image. The basic and the multi-scale gradient operations are explained. The concept of the multi-scale gradient operator for watershed transform is also presented.

2.1 Mathematical Morphology

Mathematical morphology is concerned with the analysis of signals in terms of shape. In image processing, morphology is the name of a specific methodology for analyzing the geometric structures inherent within an image. Generally speaking, the morphological operators transform the original image into another image through the interaction with another image of a certain shape and size, which is known as the *structuring element*. Geometric features of the image, which are similar in shape and size to the structuring element, are preserved, while the other features are suppressed. Therefore, morphological operations can simplify the image data, while preserving their essential shape characteristics and eliminating irrelevances. Non-morphological approaches to image processing are typically based on linear transformations such as convolution, whereas mathematical morphology uses tools of non-linear algebra and operates with point sets, and their connectivity and shape. Morphological operations simplify images, and quantify and preserve the main characteristics of objects. Morphological operations are used predominantly for the following purposes:

- Image pre-processing (noise filtering, shape simplification with alternate sequential filter).

- Enhancing object structure (skeletonisation, thinning, convex hull, object masking)
- Extracting objects from the background (top-hat transformation, watershed, reconstruction)
- Quantitative description of object (area, perimeter, projections)

A systematic introduction to the theoretical foundations of mathematical morphology, its main image operations, and their applications can be found in [GD88, Ser82]. Mathematical morphology exploits the point set properties, and the results of integral geometry and topology. The initial assumption states that the real images can be modelled using the point sets of many dimensions for example, the N-dimensional Euclidean space. Image processing uses the digital counterpart of the Euclidean sets of integer pairs ($\in Z^2$) for binary image morphology or the set of integer triples ($\in Z^3$) for grey scale morphology or binary 3-D morphology where Z denotes the set of positive integers. The morphologic operations work with two images, namely the original image to be processed and the structuring element. Each structuring element has a shape which can be thought of as a parameter to the operation. The most fundamental morphological operations are the morphological *dilation* and the morphological *erosion*. Based on these, two compound operations known as *opening* and *closing* are defined. The basic definitions of the binary morphologic operations and those related to the gray scale morphology are given in the following subsections.

2.1.1 Binary Morphology

The theoretical foundation of binary mathematical morphology lies in set theory. Of the entire set of pixels constituting a binary image, those with value 1 may be denoted as the foreground pixels, while the set of those with 0 value may be referred to as the background. Let the set A represent the original binary image and B be the set of points ($\in Z^2$) representing the binary one pixels of the structuring element.

Basics Definitions

Definition 2.1 (Translation) *Given an image A, the translation of A by the point x, denoted by $(A)_x$, is defined as*

$$(A)_x = \{c | c = a + x, \text{ for } a \in A\} \tag{2.1.1}$$

Definition 2.2 (Reflection) *The reflection of B, denoted by $\hat{B}$, is defined as*

$$\hat{B} = \{x | x = -b, \text{ for } b \in B\} \tag{2.1.2}$$

Definition 2.3 (Complement) *The complement of set A is defined as*

$$A^c = \{x | x \notin A\} \tag{2.1.3}$$

Definition 2.4 (Difference) *The difference of two sets A and B, denoted by $A - B$, is defined as*

$$A - B = \{x | x \in A,\ x \notin B\} \tag{2.1.4}$$

Binary Dilation

Definition 2.5 (Dilation) *Dilation of a binary image A by a binary structuring element B, denoted by $A \oplus B$, is defined as*

$$A \oplus B = \{w \in Z^2 | \, w = a + b, \ \text{where } a \in A \text{ and } b \in B\} \tag{2.1.5}$$

Based on (2.1.1), the above equation (2.1.5) can be rewritten as

$$A \oplus B = \{x | (\hat{B})_x \cap A \neq \varnothing\} \tag{2.1.6}$$

Dilation is found by placing the center of the structuring element over each of the foreground pixels of the original image and then taking the union of all the resulting copies of the structuring element, produced by using the translation operation. Dilation modifies the original image with respect to the shape of the structuring element. In general, dilation has the effect of "expanding" an image and hence, the small holes can be eliminated from the original image in the process.

Binary Erosion

Definition 2.6 (Erosion) *Erosion of a binary image A by a binary structuring element B, denoted by $A \ominus B$, is defined as*

$$A \ominus B = \{p \in Z^2 | p + b \in A, \ \forall b \in B\} \tag{2.1.7}$$

The above equation (2.1.7) can be rewritten as

$$A \ominus B = \{x | (B)_x \subseteq A\} \tag{2.1.8}$$

Like dilation, the erosion of the original image by the structuring element can be described intuitively by translation. The structuring element is moved across the original image. For a given foreground pixel, one puts the center of the structuring element onto it, that is, one translates the structuring element to that pixel. If the translation of the structuring element is a subimage of the original image, then that pixel is activated in erosion; otherwise it is eliminated. In general, erosion "shrinks" the original image and the small peaks are eliminated from the original image in the process.

Binary Opening

Definition 2.7 (Opening) *Opening of a binary image A by a binary structuring element B, denoted by $A \circ B$, is defined as*

$$A \circ B = (A \ominus B) \oplus B \tag{2.1.9}$$

From the above definition, the original image A is eroded first by B and then dilated by B. Therefore, it can be intuitively thought as "rolling the structuring element about the inside boundary of the image". The following equation gives a rigorous set-theoretic characterization of this "fitting" property. It states that the opening of A by B is obtained by taking the union of all translations of B which fit into A.

$$A \circ B = \bigcup \{(B)_x | (B)_x \subseteq A\} \tag{2.1.10}$$

Opening generally smoothes the contour of an image, breaks narrow isthmuses, and eliminates thin protrusions. Figure 2.1 shows how the image is smoothed and the spot-like noise is removed because the disk cannot fit into them. It should be noted that the smoothing effect of the object boundary highly depends on the shape of the structuring element.

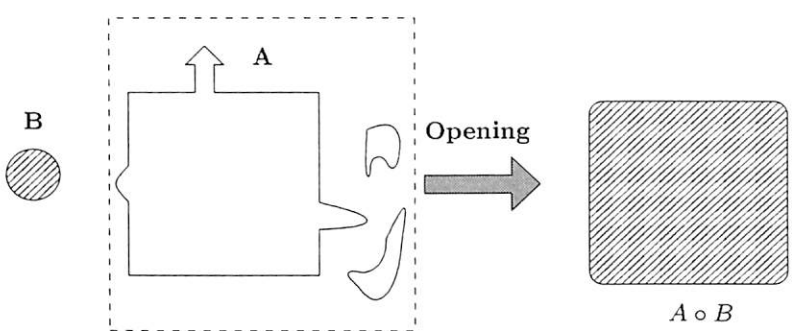


Figure 2.1: Opening

Binary Closing

Definition 2.8 (Closing) *Closing of a binary image A by the structuring element B, denoted by $A \bullet B$, is defined as*

$$A \bullet B = (A \oplus B) \ominus C \tag{2.1.11}$$

As depicted in (2.1.11), closing is done by first dilating the image and then eroding it. The closing of the original image includes all the points satisfying the condition stated in (2.1.11); that is, whenever

the point can be covered by a translation of the structuring element, there must be some point in common between the translated structuring element and the original image. Geometrically, a point z is an element of $A \bullet B$ if and only if $(B)_x \cap A \neq \oslash$ for any translation $(B)_x$ of B that contains z. Closing also tends to the smooth the sections of the contours but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes, and fills the gaps in the contour. Hence, instead of eliminating the small peaks, it will "fill" the holes, as shown in Figure 2.2

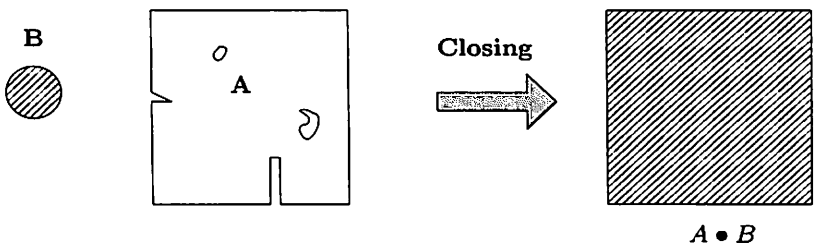


Figure 2.2: Closing

2.1.2 Gray-Scale Morphology

A 2D gray-scale image with gray-level function $g(r, c)$ where r and c are spatial coordinates, can be thought of as a set of points $p = (r, c, g(r, c))$ in the Euclidean three-dimensional (3D) space. By applying the *umbra transform* [Ste86, HSZ87] U on $g(r, c)$, a 2D gray-scale image can be transformed into a 3D binary image. Therefore, the gray-scale morphological operators may be regarded as the extension of the binary morphological operators to the three-dimensional space.

Definition 2.9 (Umbra Transform) *Given a signal f, the umbra transform of f, denoted by $U[f]$, is defined as*

$$U[f] = \{(x, y) : x \in D_f \text{ and } y \leq f(x)\} \tag{2.1.12}$$

where D_f : domain of the signal f

Let s denote the coordinate of the pixels as one traverses an image from left to right along the first row, then along the second row, and so on. Fig. 2.3(a) depicts the gray level values of a typical image f corresponding to the different values of s. Fig. 2.3(b) shows how the umbra transform of the signal f is generated.

Definition 2.10 (Top Surface) *Given the umbra U of a signal f, the top surface of U is defined as*

$$T[U[f]] = \{(x, y) | x \in D_f \text{ and } y = max\{y | (x, y) \in U\}\} \tag{2.1.13}$$

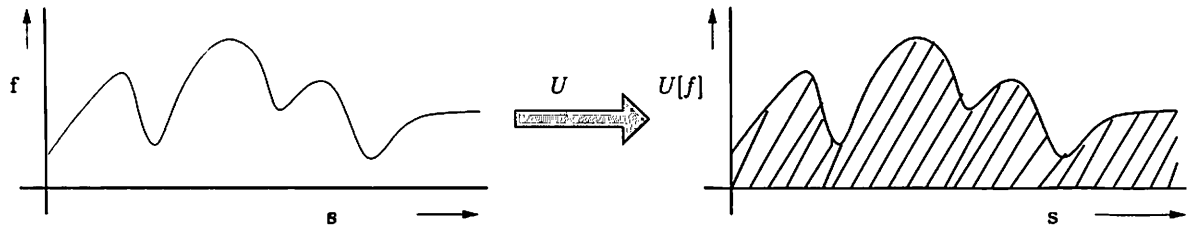


Figure 2.3: The top and umbra transform

Thus, the top and the umbra are inverse operators, that is,

$$T[U[f]] = f$$

Gray-Scale Dilation

Definition 2.11 (Gray-scale Dilation) *The dilation of a gray-scale image $f(r, c)$ by a gray-scale structuring element $g(r, c)$ is denoted by $(f \oplus g)(r, c)$ or $\delta_g(f)(r, c)$ and is defined as*

$$f \oplus g = T\{U[f] \oplus U[g]\} \tag{2.1.14}$$

It is the 'binary dilation' of the two umbra functions. Equation (2.1.14) can be re-written as

$$\delta_g(f)(r, c) = max_{(i,j)}\{f(r - i, c - j) + g(i, j) | ((r - i), (c - j)) \in D_f;\ (i, j) \in D_g\} \tag{2.1.15}$$

Where D_f and D_g are the domains of f and g respectively. Dilation of a gray-scale image is the operation of assigning to each pixel the maximum value found over a neighborhood of the correspondin; pixel in the input image. As before, g is the structuring element of the morphological process; however, g is now a function rather than a set. In the gray-scale case, the structuring element is a function of two variables that specifies the desired local gray-level property. The value of the structuring element is subtracted from the original image when the maximum is calculated in the neighborhood. Since dilation is commutative, the operation $g \oplus f$ can be performed by interchanging f and g in (2.1.14). The result is the same, and g is now the function translated.

Gray-Scale Erosion

Definition 2.12 (Gray-scale Erosion) *The erosion of a gray-scale image $f(r, c)$ by a gray-scale structuring element $g(r, c)$ is denoted by $(f \ominus g)(r, c)$ or $\varepsilon_g(f)(r, c)$ and is defined as*

$$f \ominus g = T\{U[f] \ominus U[g]\} \tag{2.1.16}$$

It is the 'binary erosion' of the two umbra functions. Equation (2.1.16) can be re-written as

$$\varepsilon_g(f)(r,c) = min_{(i,j)}\{f(r+i,\ c+j) + g(i,\ j) | ((r+i),\ (c+j)) \in D_f;\ (i,j) \in D_g\} \quad (2.1.17)$$

where D_f and D_g are the domains of f and g respectively. Erosion of a gray-scale image is the operation of assigning to each pixel the minimum value found over a neighborhood of the corresponding pixel in the input image. The value of the structuring element is added when the minimum is calculated in the neighborhood. It should be noted that the property of erosion (in contrast to dilation) is not commutative.

Figure 2.4(a) shows the original 176×144 Akiyo gray-scale image. Fig. 2.4(b) shows the result of dilating the original image with a "flat-top" 3×3 structuring element. The dilated image, in which the small dark regions are reduced, is brighter than the original image. Figure 2.4(c) shows the result of eroding the original image. The eroded image is darker, and the sizes of the small, bright features are reduced. Gray-scale opening and closing are often used in applications to extract the parts of a

(a) (b) (c)

Figure 2.4: (a) Original akiyo image; (b) result of dilation; (c) result of erosion.

gray-scale image with a given shape and a gray-scale structure.

Gray-scale Opening and Closing

Definition 2.13 (Gray-scale Opening) *The opening of a gray-scale image $f(r,c)$ by a gray-scale structuring element $g(r,c)$ is denoted by $(f \circ g)(r,c)$ or $\gamma_g(f)(r,c)$ and is defined as*

$$\begin{aligned} \gamma_g(f)(r,c) &= (f \ominus g) \oplus g \\ &= \delta_g(\varepsilon_g(f)) \end{aligned} \quad (2.1.18)$$

Definition 2.14 (Gray-scale Closing) *The closing of a gray-scale image $f(r,c)$ by a gray-scale structuring element $g(r,c)$ is denoted by $(f \bullet g)(r,c)$ or $\phi_g(f)(r,c)$ and is defined as*

$$\begin{aligned} \phi_g(f)(r,c) &= (f \oplus g) \ominus g \\ &= \varepsilon_g(\delta_g(f)) \end{aligned} \tag{2.1.19}$$

The opening and closing for gray-scale images are morphological duals with respect to complementation and reflection. That is,

$$(f \bullet g)^c = f^c \circ \hat{g}. \tag{2.1.20}$$

Where $f^c = -f(r,c)$ and $\hat{g} = g(-r,-c)$.

Figure 2.5(b) shows the result of opening the original image with a 3×3 "flat-top" structuring element. The bright details in Fig. 2.5(a) which are smaller in size than the structuring element are eliminated, with no appreciable effect on the darker gray levels. Figure 2.5(c) shows the result of closing of Fig. 2.5(a). The smaller dark details of the original image are eliminated, with relatively little effect on the bright features.

(a) (b) (c)

Figure 2.5: (a) Original Akiyo image; (b) result of opening; (c) result of closing.

2.2 Morphological Gradient Operators

Gradient operators are used in image segmentation because they highlight the intensity variations in the images. These variations are assumed to be the edges of the objects in an image. Hence, the conventional approach to edge detection is composed of "gradient computation" and "thresholding". First, the original image is transformed into a gradient image which represents the edge strength of each pixel and then, a threshold is applied to classify each pixel to an edge point or a non-edge point.

Traditionally, the gradient image can be obtained by means of the first order differential operators or the Laplacian operators which can enhance the spatial intensity changes in the image. Some of the most popular gradient operators based on the linear transformations and template-matching are Prewitt [GW01], Sobel [Dav75] and Canny [Can86] operators. Morphological edge detectors have also been proposed for their robustness under noisy condition and some of them are discussed in the following. Morphological gradient is based on the concept of *residues* which is derived by means of 'difference' operations involving the transformations or filters. Definitions of some simple morphological gradient operators are given as follows [Sal94]:

Gradient by Erosion: The internal gradient $G^-(f)$ or $G_e(f)$ is defined as the difference between the original image f and the eroded image:

$$G_e(f) = f - (f \ominus B) = f - \varepsilon_B(f)$$

Gradient by Dilation: The external gradient $G^+(f)$ or $G_d(f)$ is defined as the difference between the dilated image and the original image:

$$G_d(f) = (f \oplus B) - f = \delta_B(f) - f$$

Morphological Gradient: The basic morphological gradient $G(f)$, also called Beucher's gradient [Beu90a], is defined as the arithmetic difference between the dilation and the erosion with the elementary structuring element 'B' of the considered grid:

$$\begin{aligned} G(f) &= (f \oplus B) - (f \ominus B) \\ &= \delta_B(f) - \varepsilon_B(f) \end{aligned} \tag{2.2.1}$$

This gradient is referred to as the mono-scale morphological gradient operator.

If B is chosen as the *rod* structure element with *flat top* whose domain is the 4-neighbor of the origin including it (see Fig. 2.6), then the gradients by erosion and dilation are also called the *erosion residue edge detector* and the *dilation residue edge detector* respectively. Figures 2.7(a), 2.7(b) and 2.7(c) show the result of performing the morphological gradient by dilation, gradient by erosion and Beucher's gradient respectively of the image shown in 2.4(a) with the D_{rod1} structuring element (from Fig. 2.6). However, the above three morphological residue gradients are not good enough due to their sensitivity to noise. Their performance depends on the size of the structuring element. If the structuring

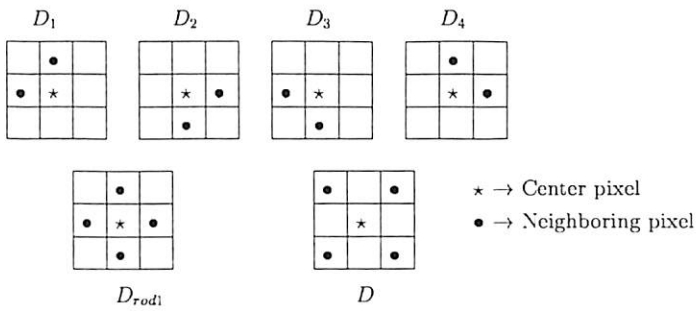


Figure 2.6: Domains of the 3×3 structuring elements.

(a) (b) (c)

Figure 2.7: (a) result of gradient by dilation; (b) result of gradient by erosion; (c) result of morphological gradient.

element 'B' is large, then the output of the gradient operator for ramp edges is equal to the edge height which allows more noise to be eliminated. Unfortunately, large structuring elements result in serious interaction among the edges, which may lead to gradient maxima not coinciding with the edges (thus producing thicker edges). If the structuring element is very small, this gradient operator may detect the fine edges but becomes more sensitive to noise.

In order to exploit the advantages of both the large and the small structuring elements, multi-scale gradient operation is introduced. A multi-scale gradient operator obtained by combining the morphological gradients in different scales is not only insensitive to noise but also it extracts the various fineness of the edges. A typical multi-scale morphological operation based on the improved *erosion residue* and the *dilation residue* operators has been introduced by B. Chanda *et al* [CKP98]. First, the improved erosion residue operator and the dilation residue operator are defined as

$$G_e^{'}(f) = min\{f - \varepsilon_{D_{rod1}}(f),\ f - \varepsilon_D(f),\ G_e^{''}(f)\} \tag{2.2.2}$$

where $G_e^{''}(f)$ is defined as

$$G_e^{''}(f) = max\{\varepsilon_{D_1}(f) - \varepsilon_{D_2}(f),\ \varepsilon_{D_3}(f) - \varepsilon_{D_4}(f)\} \tag{2.2.3}$$

and

$$G'_d(f) = min\{\delta_{D_{rod1}}(f) - f,\ \delta_D(f) - f,\ G''_d(f)\} \tag{2.2.4}$$

where $G''_d(f)$ is defined as

$$G''_d(f) = max\{\delta_{D_1}(f) - \delta_{D_2}(f),\ \delta_{D_3}(f) - \delta_{D_4}(f)\} \tag{2.2.5}$$

respectively. Then, the new edge detector G'_{sum} is defined as

$$G'_{sum} = G'_d(f) + G'_e(f) \tag{2.2.6}$$

In Fig. 2.6, each structuring element is defined with radius 1. To extend the above operators to scale n, where n is a positive integer, the structuring elements are modified in proportion to n. That is, nD_1 nD_2, nD_3 and nD_4 have the same domain as D_1, D_2, D_3, and D_4 but have size n. Similarly, $D_{rodn} \triangleq nD_{rod}$ and $nD = nN_8/nN_4 \cup (0,0)$ (where N_8 means 8-connected neighbor, N_4 means 4-connected neighbor and '/' denotes the set difference operator). Therefore, the dilation residue operator at scale n is defined as

$$G^n_d(f) = min\{\delta_{D_{rodn}}(f) - f,\ \delta_{nD}(f) - f,\ G^{n'}_d(f)\} \tag{2.2.7}$$

where $G^{n'}_d(f)$ is defined as

$$G^{n'}_d(f) = max\{\delta_{nD_1}(f) - \delta_{nD_2}(f),\ \delta_{nD_3}(f) - \delta_{nD_4}(f)\} \tag{2.2.8}$$

The improved erosion residue operator at scale n can be defined in a similar way. Suppose the edge strength at scale n due to an image f is denoted by f'_n. After defining the edge image of each scale, one has to combine the edge strengths. It is intuitive to use the weighted pixel summation, i.e.

$$f' = \sum_{i=k}^{l} w_i f'_i \tag{2.2.9}$$

where $[k, l]$ represents the range of the scales and w_i's are the respective weights of the different scales. Once the edge strength of a pixel has been found out, one can directly threshold each pixel to classify it into an edge or a non-edge. It makes use of "*non-maximal suppression*". Non-maximal suppression [Can86] nullifies the gradient value at a pixel if its gradient magnitude is non-maximal among the neighboring magnitudes along the perpendicular to the gradient orientation of the pixels. The result is a thinned edge map, where only the dominant edges are present. Note that the combined edge image appears to contain the range of mountains and the true edge lies along its ridge. The authors of [CKP98]

conclude that the multi-scale gradient operator given by (2.2.7) gives a better noise immunity and orientational and positional response compared to most of the conventional morphological operators. Also, it is computationally less expensive than the conventional edge detectors like Canny [Can86], but with comparable result. However, the gradient operation described by (2.2.7) sometimes produces low gradient values. This operation also produces too many irrelevant contours if a large number of local minima are present in the input image.

Alternatively, a multi-scale gradient proposed by Wang [Wan97] has been found to be an important gradient operator for watershed transform computation method, because it produces a continuous closed contour of an object and controls the over-segmentation. The next section presents a detailed description of this multi-scale gradient operator which will be shown to be a basis for building up the edge image that is suitable for watershed transformation.

2.2.1 Multi-scale Morphological Gradient Operator for Watershed Transform Computation

The ideal gradient operator for watershed transformation is the one whose output is equal to or greater than the input edge height, but not the edge slope. However, for ramp edges, the conventional gradient algorithms produce the value equal to the edge slope. When a gradient image with lower intensity values is given as an input to the watershed algorithm, then an overwhelming number of contours result, including the unimportant ones. Hence, conventional gradient algorithms, such as Gaussian filter, Laplacian operator and some morphological gradient operators, produce too many local minima in the homogeneous regions due to noise or quantization error. This leads to partitioning a homogeneous region into a large number of "spots". The most straightforward methodology is thresholding the gradient directly. However, a conventional gradient operator produces low gradient values at blurred edges, even though the intensity change between the two sides of an edge may be high. By thresholding, one can eliminate the local minima caused by noise and quantization error while preserving those produced by blurred edges. Another solution to this problem is to extract the markers and impose them on the gradient image. This may require prior information about the objects and background to be segmented. After watershed transformation, region merging or relaxation labelling is usually performed to further remove the false contours. This process may be computationally much more expensive than the watershed transformation because it requires merging numerous catchment basins.

To overcome the problem stated above, a multi-scale gradient algorithm based on morphological

operators followed by a procedure of local minima elimination has been proposed by Wang [Wan97]. Some useful operations are defined next, followed by the detailed description of Wang's multi-scale gradient algorithm.

Geodesic transforms [LM84, Vin93] of a function f are defined with respect to a reference function g. The geodesic dilation of size one is defined as the minimum of the dilation of size one of original function f and the reference function g. The geodesic dilation (or erosion) of size n, also called reconstruction by dilation (or by erosion) can be defined as follows:

Definition 2.15 (Dilation-based Grayscale Reconstruction) *Let f and g be two grayscale images defined on the same domain and $g \leq f$. The grayscale reconstruction $\gamma_f^{(rec)}(g)$ of f from g is obtained by iterating the elementary geodesic dilations of g "under" f until stability is reached.*

$$\gamma_f^{(rec)}(g) = \bigvee_{n \geq 1} \delta_f^{(n)}(g) \tag{2.2.10}$$

where $\delta_f^{(n)}(g)$ can be obtained by iterating n elementary geodesic dilation operations

$$\delta_f^{(n)}(g) = \underbrace{\delta_f^{(1)}(\delta_f^{(1)}(\cdots \delta_f^{(1)}(g)))}_{n \text{ times}}$$

and the geodesic dilation is defined as

$$\delta_f^{(1)} = (g \oplus B) \wedge f$$

(where B is the flat structuring element of size 1 and $\wedge$ stands for pointwise minimum.)

Definition 2.16 (Erosion-based Grayscale Reconstruction) *Let f and g be two grayscale images defined on the same domain and $f \leq g$. The grayscale reconstruction $\phi_f^{(rec)}(g)$ of f from g is obtained by iterating the elementary geodesic erosions of g "above" f until stability is reached.*

$$\phi_f^{(rec)}(g) = \bigwedge_{n \geq 1} \varepsilon_f^{(n)}(g) \tag{2.2.11}$$

where $\varepsilon_f^{(n)}(g)$ can be obtained by iterating n elementary geodesic erosion operations

$$\varepsilon_f^{(n)}(g) = \underbrace{\varepsilon_f^{(1)}(\varepsilon_f^{(1)}(\cdots(\varepsilon_f^{(1)}(g))))}_{n \text{ times}}$$

and the geodesic erosion is defined as

$$\varepsilon_f^{(1)}(g) = (g \ominus B) \vee f$$

(where B is the flat structuring element of size 1 and $\vee$ stands for pointwise maximum.)

Better contour preservation can be achieved by the following reconstruction techniques.

Definition 2.17 (Opening by Reconstruction of Erosion) *Let f be a grayscale image and $\varepsilon(f)$ be the eroded image with flat structuring element B, and $\varepsilon(f) \leq f$. The opening by reconstruction of erosion is defined as*

$$\gamma^{(rec)}(\varepsilon(f), f) = \bigvee_{n \geq 1} \delta_f^{(n)}(\varepsilon(f)) \tag{2.2.12}$$

where $\delta_f^{(n)}(\varepsilon(f))$ can be obtained by iterating n elementary geodesic dilations

$$\delta_f^{(n)}(\varepsilon(f)) = \underbrace{\delta_f^{(1)}(\delta_f^{(1)}(\cdots(\delta_f^{(1)}(\varepsilon(f)))))}_{n \text{ times}}$$

and the geodesic dilation is defined as

$$\delta_f^{(1)}(\varepsilon(f)) = (\varepsilon(f) \oplus B) \wedge f$$

(where B is the flat structuring element of size 1 and $\wedge$ stands for pointwise minimum.)

Definition 2.18 (Closing by Reconstruction of Dilation) *Let f be a grayscale image and $\delta(f)$ be the dilated image with flat structuring element B, and $f \leq \delta(f)$. The opening by reconstruction of dilation is defined as*

$$\phi^{(rec)}(\delta(f), f) = \bigwedge_{n \geq 1} \varepsilon_f^{(n)}(\delta(f)) \tag{2.2.13}$$

where $\varepsilon_f^{(n)}(\delta(f))$ can be obtained by iterating n elementary geodesic erosions

$$\varepsilon_f^{(n)}(\delta(f)) = \underbrace{\varepsilon_f^{(1)}(\varepsilon_f^{(1)}(\cdots(\varepsilon_f^{(1)}(\delta(f)))))}_{n \text{ times}}$$

and the geodesic dilation is defined as

$$\varepsilon_f^{(1)}(\delta(f)) = (\delta(f) \ominus B) \vee f$$

(where B is the flat structuring element of size 1 and $\vee$ stands for pointwise maximum.)

In the case of (2.2.12), image simplification is performed by the opening operation γ, which eliminates all the components smaller than the structuring element. The reconstruction restores the contour of the components that have not been totally removed by the opening. Based on (2.2.12) and (2.2.13), one can define open-close by reconstruction and close-open by reconstruction operations as follows.

Definition 2.19 (Open-Close by Reconstruction) *Let f be a grayscale image. The open-close by reconstruction operation is defined as opening by reconstruction followed by closing by reconstruction.*

$$\phi^{(rec)}(\delta(\gamma^{(rec)}(\varepsilon(f), f)), \gamma^{(rec)}(\varepsilon(f), f)) \tag{2.2.14}$$

Definition 2.20 (Close-Open by Reconstruction) *Let f be a grayscale image. The close-open by reconstruction operation is defined as closing by reconstruction followed by opening by reconstruction.*

$$\gamma^{(rec)}(\varepsilon(\phi^{(rec)}(\delta(f), f)), \phi^{(rec)}(\delta(f), f)) \tag{2.2.15}$$

It has to be mentioned that the practical implementation of these functions can be done by very efficient techniques relying on waiting list structures which avoid any iterative process and lead to extremely fast algorithms [Vin90]. The above reconstructions are shown for the Akiyo image (176×144) in Figures 2.8 – 2.10.

(a) (b) (c)

Figure 2.8: (a) Original image; (b) result of opening; (c) result of opening by reconstruction.

(a) (b) (c)

Figure 2.9: (a) Original image; (b) result of closing; (c) result of closing by reconstruction.

The contour preservation capability of these operations is demonstrated by these reconstruction results. To show the contour preservation in presence of noise, zero mean Gaussian noise of variance 0.01 is added to the original image. The reconstruction of the noisy image by open-close by reconstruction operation is shown in Figure 2.11(c). It exhibits contour preservation while smoothing the noise in the homogeneous regions.

A multi-scale gradient image [Wan97] is used as input to the watershed transform algorithms in the present work. The multi-scale gradient operation is performed by the following three steps.

(a) (b) (c)

Figure 2.10: (a) Original image; (b) result of open-close (c) result of open-close by reconstruction.

(a) (b) (c)

Figure 2.11: (a) After adding Gaussian noise (μ=0, σ=0.01) to the original Akiyo image; (b) result of open-close of noise contaminated image; (c) result of open-close by reconstruction of noise contaminated image

Multi-scale gradient operator [Wan97]

Let f be a grayscale image and h be a given threshold value.

1. *Pre-filtering:* In order to eliminate the noise disturbance, the original image f is pre-filtered using the open-close by reconstruction operation given by (2.2.14) with a flat-top structuring element consisting of 3×3 pixels.

2. *Multi-scale Morphological Gradient:* Let $f^{'}$ be the prefiltered grayscale image and B_i a group of square structuring elements. The multi-scale gradient is defined as

$$MG(f^{'}) = \frac{1}{n}\sum_{i=1}^{n}[((f^{'} \oplus B_i) - (f^{'} \ominus B_i)) \ominus B_{i-1}] \qquad (2.2.16)$$

where n is the scale, and the size of B_i is $(2i+1)\times(2i+1)$ pixels. The choice of this structuring element has the advantage of low computational cost. Moreover, other structuring elements like the line segments with specific directions, can be used for specific applications.

3. *Elimination of Small Local Minima:* Local minima consist of a small number of pixels or have low contrast due to noise or quantization error. Given the gradient image by applying the multi-scale gradient stated in the last step, the procedure of local minima elimination in [Wan97] can be stated as

   (a) The gradient image $MG(f')$ is dilated with a square structuring element B_s made of 2×2 pixels.

   (b) A constant h is added to the dilated gradient image, and then one performs the erosion-based grayscale reconstruction given by (2.2.11). Hence, the final gradient can be expressed as

$$\phi^{(rec)}_{(MG(f'))}((MG(f') \oplus B_s) + h) \tag{2.2.17}$$

In step (a), if the local minima is "norrower" than the size of B_s, it will be filled by the process of dilation. In step (b), the local minima with a contrast lower than h can be filled irrespective of their absolute values. If the local minimum is wide and deep, it cannot be removed. As h increases, the number of regions produced decreases. Figure 2.12 shows that an increase in h leads to a reduction in the number of regions.

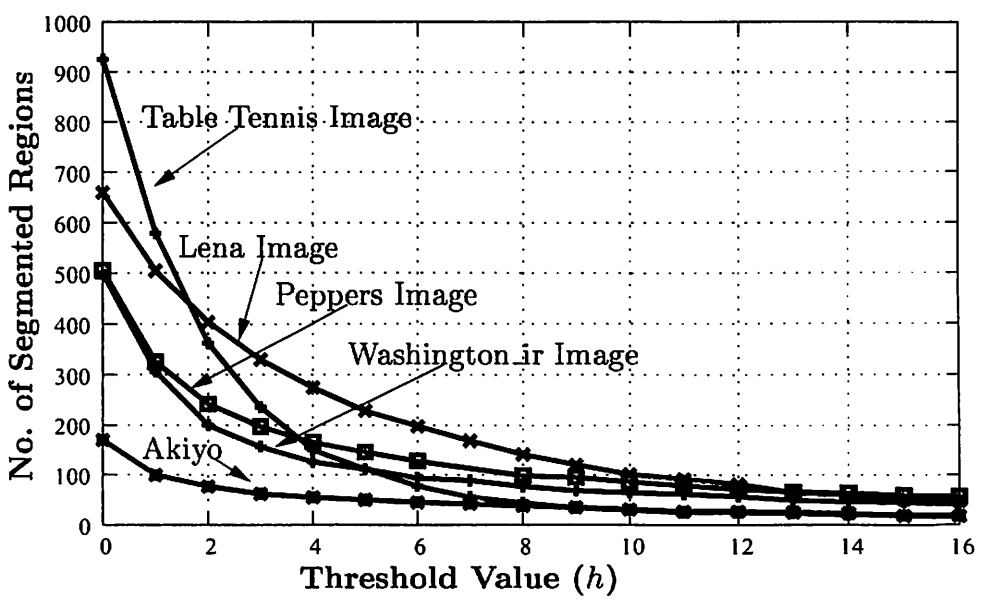
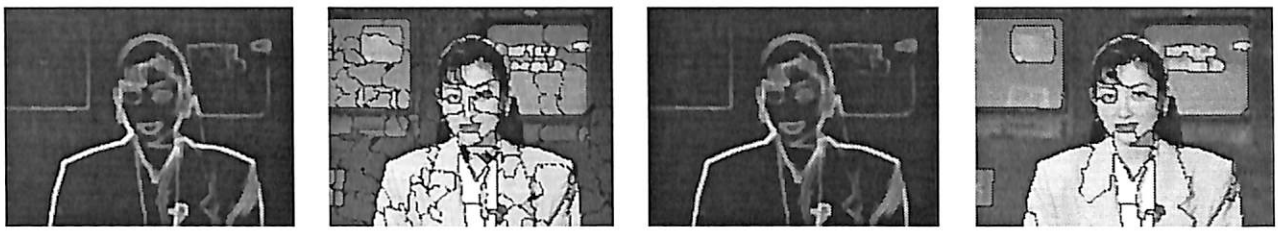


Figure 2.12: Number of regions for different threshold value (h).

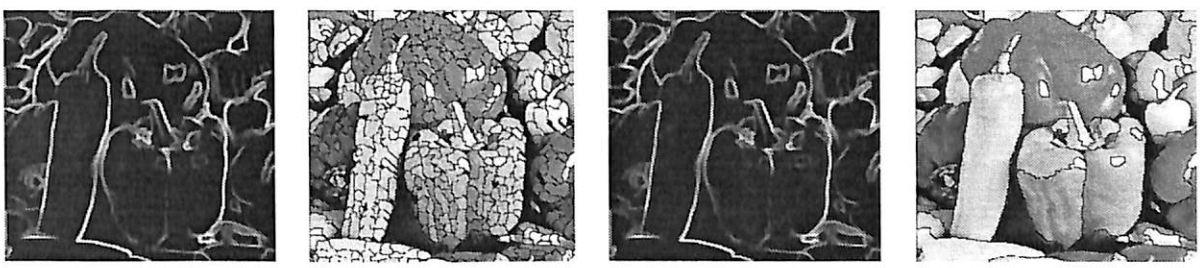
However, compared to the operation of directly thresholding the gradient image, which removes only the minima with low absolute value, it is more reasonable to use the morphological approach shown

above to eliminate the local minima. The outputs of the watershed algorithm considering the multi-scale gradient algorithm [Wan97] are depicted in Figures 2.13 and 2.14. Two different values of h are considered.



(a) Multi-scale gradient image (h=0) (b) Segmented image(No. of regions = 170) (c) Multi-scale gradient image (h=10) (d) Segmented image (No. of regions =31)

Figure 2.13: The output of the multi-scale gradient operation and the watershed transformation [VS91] with threshold h=0 and 10 of the original Akiyo image (176×144).



(a) Multi-scale gradient image (h=0) (b) Segmented image(No. of regions = 540) (c) Multi-scale gradient image (h=10) (d) Segmented image (No. of regions =85)

Figure 2.14: The output of the multi-scale gradient operation and the watershed transformation [VS91] with threshold h=0 and 10 of the original Peppers image (256 × 256).

2.3 Conclusions

In this chapter, a systematic study has been carried out on the multi-scale gradient operator which is often used during the pre-processing step of image segmentation by the watershed algorithm. Multi-scale filters on the basis of mathematical morphology are robust to noise compared to a traditional first-order difference operator or the Laplacian operator. The conventional operators usually produce over-segmentation because of generation of too many local minima. To avoid this problem, an appropriate means for pruning the overabundance of local minima is needed [Wan97]. Experimental

results [CKP98, Wan97] indicate that the watershed transform with Wang's multi-scale gradient algorithm [Wan97] produces meaningful contours of homogeneous regions. Chanda's algorithm [CKP98] based on multi-scale directional gradient, however, does not yield the desired result, that is, the elimination of local minima is not very effectively achieved. On the other hand, Wang's method [Wan97] achieves suppression of the overwhelming number of local minima by using dilation operation with a structuring element of size (2×2) and thresholding operation for local minima containment.

Chapter 3

An Improved Flooding-based Watershed Algorithm and its Prototype Architecture

This chapter introduces a flooding-based watershed algorithm which has been derived by improving upon a standard algorithm [BM93] due to Meyer. Section 3.1 discusses about the standard algorithm based on ordered queue. The next section deals in depth with the proposed improved algorithm. The analysis of the proposed algorithm is also given in this section. Section 3.3 presents the results of executing the proposed algorithm as well as Meyer's algorithm on a number of standard images. In addition, this section evaluates the quality of segmentation ensured by the proposed algorithm in a quantitative manner. Section 3.4 is concerned with developing a hardware architecture for the improved algorithm and its realization in FPGA (Field Programmable Gate Array) Technology.

3.1 Meyer's Watershed Algorithm

The sequential watershed transform method based on ordered queue has been proposed by Meyer [BM93], in which the flooding of the water in the topographic surface is simulated using H FIFO queues, $H \leq 255$. Some useful notations are defined next, followed by a detailed description of the ordered queue-based watershed algorithm [BM93].

3.1.1 Basic Definitions

Let us consider a 2-D digital gray scale image F whose domain is denoted as $D_F \subset \mathbb{Z}^2$ and $F : D_F \rightarrow \mathbb{N}$ is a function assigning an integer value to each $p \in D_F$. For each pixel $p \in D_F$, $F(p)$ is called the gray value or altitude of p (considering F as a topographical surface). Let G denote the underlying

digital grid, which can be a square grid (with *4-connectivity* or *8-connectivity*) or a hexagonal grid (with *6-connectivity*). Moreover, G is a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$. The set $N_G(p)$ is made of the neighbors of a pixel p on the grid G.

Definition 3.1 (Isolated Minimum) *A pixel $p \in D_F$ is called an isolated minimum, if $F(p) < F(q)$, $\forall q \in N_G(p)$.*

Definition 3.2 (Plateau) *A pixel $p \in D_F$ is said to lie on a plateau P with altitude h in an image F, if $\exists q \in N_G(p)$ with $h = F(p) = F(q)$.*

Definition 3.3 (Outer Pixel) *A pixel $p \in D_F$ is called an* outer *pixel of a plateau P, if p is on the plateau P and $\exists q \in N_G(p)$ such that, $F(p) > F(q)$.*

Definition 3.4 (Inner Pixel) *A pixel $p \in D_F$ is called an* inner *pixel of a plateau P, if $\forall q \in N_G(p)$, $F(p) \leq F(q)$.*

Based on the above definitions, one can now define the plateau of minima and non-minima respectively.

Definition 3.5 (Plateau of Minima) *A plateau of minima (or P_M) is a connected plateau of inner pixels only. If the plateau contains a single pixel p such that $F(p) < F(q)$, $\forall q \in N_G(p)$, it is also a plateau of one (isolated) minimum.*

Definition 3.6 (Plateau of Non-minima) *A plateau of non-minima (or P_N) is a connected plateau which has one or more outer pixels and at least one inner pixel.*

The sequential watershed algorithm [BM93] based on an ordered queue consists of two main stages: regional minima detection and simulated flooding or label propagation. The algorithm starts by detecting and labelling the regions of interest. The latter are defined as connected plateaus of pixels in the gradient image which do not have neighboring pixels of lower gray level (plateau of minima). The reginal minima are identified by distinct labels. Starting from the minima, a recursive procedure of label propagation is performed. Thus, the non-labeled pixels are assimilated into different components in an increasing order of gray levels. Inside the flat areas which are not yet labeled, called plateaus of non-minima, components progress synchronously, such that they incorporate equal extents within the

plateau. The recursive label propagation is performed using an ordered queue which is in fact an array of H FIFO queues, one queue for each of the H gray levels in the image. Note that for a grayscale image, $H \leq 255$. An ordered queue naturally introduces a hierarchical ordering relation in flooding. During the flooding of a topographic surface, there appears a dual relation between the pixels, as stated below.

Property 3.1 *A pixel p acquires a label from q iff p has a higher gray value than q, or p and q have the same gray value, but q is closer to a downward brim of its plateau than p.*

The measure of closeness of a pixel within a plateau to a lower border is the geodesic distance called the *lower distance* which is defined as follows.

Definition 3.7 (Lower distance) *The lower distance d of a pixel p is as follows:*

$d(p)=0$ if pixel p is a minimum; otherwise, $d(p)$ is equal to the length s of the shortest path between two pixels p and q such that $\forall i \in \{1, 2, \ldots, s\}$, the line $(p_{i-1}, p_i) \in G$ (G being the underlying rectangular grid), $p_0 = p$, $p_s = q$ and $F(q) < F(p)$. Moreover, if $s > 1$, $\forall i \in \{1, 2, \ldots, s\text{-}1\}$, $F(p_i) = F(p_{i-1})$. $F(.)$ represents the grayscale image.

The label propagation (flooding) of Meyer's algorithm [BM93] is illustrated in Figure 3.1, in which C is the center pixel and $(N_1 - N_8)$ are its 8−connected nearest neighbors. The ordered queues (H FIFOs) are initialized with the labeled regional minima which have at least one non-labeled neighboring pixel. A pixel with gray level value of h is inserted in the h^{th} FIFO queue. The ordered queue is then scanned from the lowest gray level queue to the highest one. Pixels are dequeued from the current FIFO queue, one at a time; next, their labels are assigned to the non-labelled neighboring pixels of higher altitude, which are in turn inserted in the corresponding FIFO queues. Use of FIFO queues for serving candidate pixels within the same connected plateau ensures synchronous breadth-first propagation of labels coming from different minima inside the plateau. When all the queues have been emptied, it signifies that each pixel in the image has been labelled and the flooding procedure stops. The set of non-overlapping connected components forms a complete partition of the image and the final output image consists of all the connected components with their distinct labels. Flooding process in a sample image and its corresponding output image of labels are illustrated in Figure 3.2. A detailed description of the two major phases of Meyer's ordered queue based watershed algorithm [BM93, Mog97] is given next.

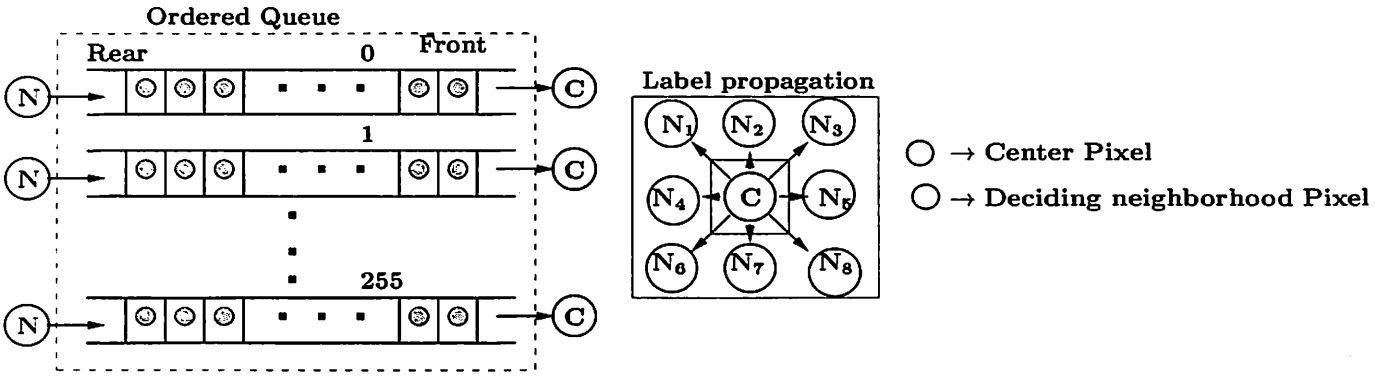


Figure 3.1: Label propagation in Meyer's algorithm.

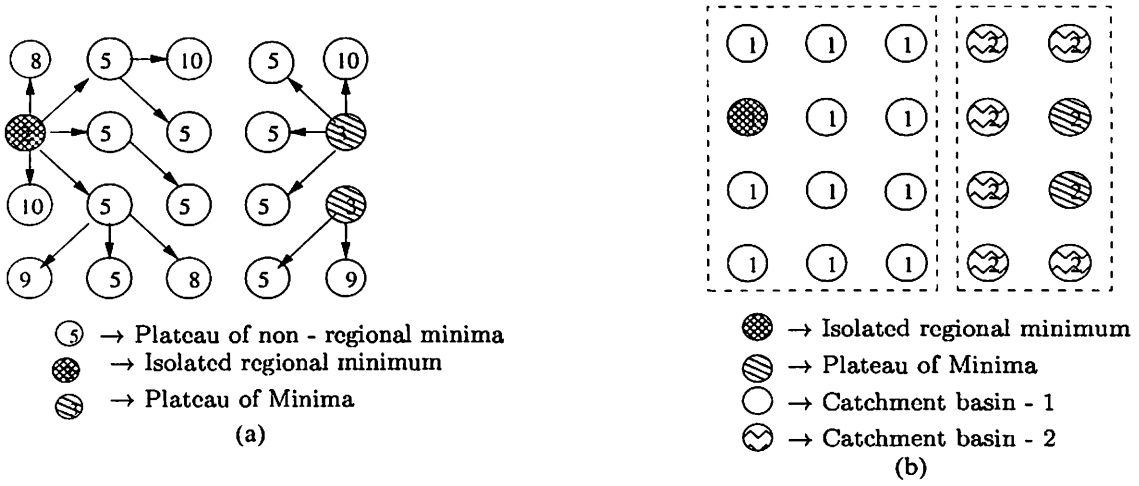


Figure 3.2: (a) Flooding the input sample image; (b) Output image of labels.

Procedure 3.1 Local Minima Detection

1: **Input**: gradient image F.
2: **Output**: label image L.
3: Initialize each pixel of the output image L to INIT.
4: **for** each pixel in the image F which is not yet visited **do**
5: **if** the pixel is a *minimum pixel* **then**
6: Assign a label to it.
7: **else if** the pixel lies on a plateau **then**
8: Check whether it is a *plateau of minima* or a *plateau of non-minima*.
9: **end if**

10: **if** the plateau is a *plateau of minima* **then**
11: Label all pixels on the plateau.
12: **else**
13: Mark all the pixels to be NARM (Not A Regional Minimum).
14: **end if**
15: **end for**
16: /* End of the procedure */

Procedure 3.2 Flooding Process

1: **Input**: gradient image F and label image L.
2: **Output**: label image L.
3: *Initialization:* An ordered queue is created with as many priority levels as there are gray levels in the image F. Initialize the ordered queue with local minima which have at least one non-labelled pixel. The local minima having gray level h goes to the h^{th} fifo queue of the ordered queue.
4: **while** the ordered queue is not empty **do**
5: Propagate the labels from the minima to the pixels of equal or higher gray level.
6: Dequeue the next pixel p from the ordered queue.
7: **for** each neighbor q of p having no label in the output image L **do**
8: Assign to q the same label as that of p in the output image L.
9: Store q in the ordered queue; its value in the image F determines its priority level in the ordered queue.
10: **end for**
11: On every non-minima plateau, labels from different sources are advanced synchronously, one step at a time.
12: **end while**
13: /* End of the procedure */

3.2 Proposed Algorithm

In Meyer's algorithm [BM93], the simulation of the flooding process involves no less than 256 queues. The proposed scheme however, uses only one queue and decides the labels of the neighboring pixels based on conditional neighborhood comparisons. The need of only a single queue instead of 256

queues makes the proposed algorithm more amenable to hardware implementation. The identification of the regional minima and their labeling in the new scheme remains the same as in Meyer's algorithm [BM93]. The label propagation, which decides the labels of the neighboring ($N_1 - N_8$) pixels of the center pixel C, requires 16 additional supporting pixels ($S_1 - S_{16}$), as illustrated in Figure 3.3.

In the first step, one detects the labelled regional minima each of which has at least one non-labelled neighboring pixel and stored in an array LM_ARRAY. Next, one initializes the FIFO queue LP_FIFO with elements from the array of labelled regional minima one at a time. Pixels are dequeued from the queue LP_FIFO, and now one attempts to decide the labels of the neighboring pixels ($N_1 - N_8$) based on the conditional neighborhood comparisons. Those pixels, whose labels can be decided, are inserted into the FIFO queue. The recursive label propagation from the present local minima stops when the FIFO queue is empty. Next, another local minimum from the array is put into the queue LP_FIFO. When all the local minima of the LM_ARRAY have been visited, it signifies that each pixel in the image has been labelled and the entire flooding procedure stops.

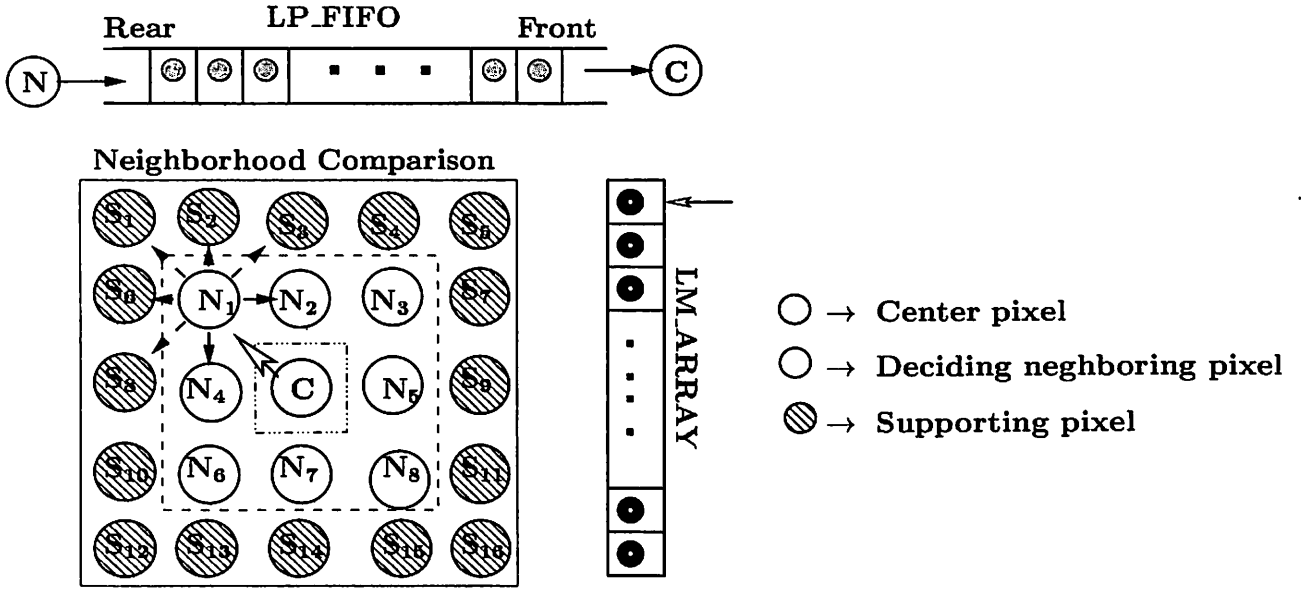


Figure 3.3: Label propagation in proposed algorithm.

To decide the label of a pixel N_i (where $i = 1, \ldots, 8$), the 4-connected neighbors of N_i (which are also the neighbors of the center pixel C) and the 8-connected supporting pixels of N_i are considered and processed in different conditional steps. For example, to find the label for N_1, at the first step we need to consider N_1 and two of its 4-connected neighbors namely N_2 and N_4 which would be compared

with the center pixel C. If the condition (namely the gray level of C being lower than or equal to that of N_2, N_4) is satisfied, then one would consider the 8-connected supporting pixels namely S_1, S_2, S_3, S_6 and S_8 which would be compared with C as shown in Figure 3.3.

Let us analyze the flooding order from the above description. A characteristic of this flooding is that waves always progress upward, or synchronized inside a plateau. As also observed in [MB90] and stated below, a dual ordering relation between pixels is imposed by flooding. Theorem 3.1, which is adopted from [Mog97] and provides the theoretical foundation of the proposed algorithm, is stated next.

Based on property 3.1 and definition 3.7, it establishes the following precedence relation between the pixels according to their flooding order.

Theorem 3.1 *$p \succ q$ (to be read p is flooded from q) in the grayscale image F whose lower distance image is d if*

$$F(q) = \min_{r \in N_G(p)} \{F(r) \mid F(r) < F(p)\}; \tag{3.2.1}$$

otherwise, $p \succ q$ if

$$\{F(p) = F(q)\} \wedge \{d(p) = d(q) + 1\}. \tag{3.2.2}$$

Proof— (proof of 3.2.1) Consider a pixel $q \in N_G(p)$ such that q satisfies the condition 3.2.1. Flooding process is known to proceed in the increasing order of the gray levels. Thus, considering flooding at level $F(q)$, it is either true that p is not yet flooded and it will be flooded by q; else, it has been already flooded from some other neighbor $r \in N_G(p)$ $(r \neq q)$. In the second case, as gray levels higher than $F(q)$ are flooded only after pixels with gray level $F(q)$ have been immersed, it must be true that $F(r) \leq F(q)$. However, as q satisfies Eq. (3.2.1), it happens to be the neighbor of p with the lowest surrounding gray level. Thus, $F(r) = F(q)$. Hence pixel p is flooded from q or r, based on which one is scanned first. Nevertheless, flooding always occurs from the surrounding neighbor with the lowest gray level, although this neighbor may not be unique.

(proof of 3.2.2) The lower distance of a pixel on a plateau may be considered to be the time taken by the wave which advances from the closest lower brim of the plateau and floods the pixel. Thus, if the pixel p on a plateau has its lower distance $d(p) > 1$, it will be flooded from a pixel which has been flooded by a wave at time $d(p) - 1$.

To prove (3.2.2) by contradiction, suppose p acquires its label from another neighbor r with lower distance $d(r) > d(p) - 1$. It signifies that the wave from r reaches p at time $d(r) + 1$ to find p unlabelled.

From the hypothesis, $d(r) + 1 > (d(p) - 1) + 1$ or $d(r) + 1 > d(p)$. Thus, the two wavefronts which reach p along pathes of shortest length have not immersed the plateau at same speed. This is however, false. Hence, (3.2.2) is proved by contradiction. □

The detailed description of the two major phases of the proposed algorithm is provided next.

3.2.1 Local Minima Detection and Labeling

In this procedure, a single scan has been employed, and a FIFO queue Q_M has been utilized. At this stage, the plateaus of minima are detected and labeled by a raster scan of the input gradient image F. The label image L is initialized with INIT value. For each pixel p which is not yet visited, its 4-connected or 8-connected neighborhood ($q \in N_G(p)$, where $N_G(p)$ is 8 or 4-connected neighbors of p) is inspected. Thus if all the neighbors are of higher gray level than p, then p is deemed to be an *isolated minimum*, and the next available label is assigned to it. Else, if p is found to have a neighbor with the same gray level as itself, it means that p is on a plateau. Next, this plateau is scanned in a breadth-first order, and the visited pixels are labelled with the current label. After scanning the plateau, if a flag named PA_Flag is found to be TRUE (meaning that a non-minima plateau is detected), then one assigns the NARM (Not A Regional Minimum) label to all the pixels forming the plateau just detected for the subsequent step of simulated flooding. The pseudocode of the local minima detection and labeling of minima is given below.

Procedure 3.3 LM_Det_Labeling(F, L) */* procedure for detecting and labeling local minima */*
Input: Morphological multi-scale gradient image F $\{F : D_F \rightarrow \mathbb{N}\}$
Output: Label image L $\{L : D_F \rightarrow \mathbb{N}\}$
Let G be a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$ and $N_G(p)$ stands for the neighbors of a pixel p on the grid G.
INIT $\leftarrow -2$; /* initialization value */
NARM $\leftarrow -1$; /* Not A Regional Minima */
$current\_label \leftarrow 0$; /* initialize the Current label */
Initialize the Label image L with INIT;
for all ($p \in D_F$ with $L(p)$ = INIT) {/* for each pixel in the image which is not yet visited during raster scan operation */} **do**
    Neighborhood_Inquiry(p, F, $pixel\_type$); /* procedure for inquiring the neighborhood pixels of p */

```
11:   if pixel_type = MINIMUM {/* it is an isolated minimum */} then
12:     L(p) ← current_label ++;
13:   else if pixel_type = ON_PLATEAU {/* a plateau is detected */} then
14:     Plateau_Analysis_Labeling(p, F, L, current_label);   /* procedure for plateau detection
        and its labeling */
15:   else if pixel_type = NON_MINIMUM {/* a non-minimum pixel is detected */} then
16:     L(p) ← NARM;
17:   end if
18: end for
19: /* End of procedure LM_det_Labeling */
```

Procedure Neighborhood_Inquiry used in the procedure LM_Det_Labeling may be described with the help of pseudo-code as follows.

```
1: Procedure 3.3.1 Neighborhood_Inquiry(p, F, pixel_type) /* procedure for neighborhood in-
   quiry */
2: Input: p, F, pixel_type.
3: Output: pixel_type.
4: pixel_type ← MINIMUM;
5: for all q ∈ N_G(p) {/* N_G(q): neighbors of p */} do
6:   if F(q) = F(p) {/* plateau detection */} then
7:     pixel_type ← ON_PLATEAU;
8:   else if F(q) < F(p) {/* non-minima detection */} then
9:     pixel_type ← NON_MINIMUM;
10:  end if
11: end for
12: /* End of procedure Neighborhood_Inquiry */
```

Next, the procedure Plateau_Analysis_Labeling used in procedure LM_Det_Labeling may be given in pseudo-code form as follows.

```
1: Procedure 3.3.2 Plateau_Analysis_Labeling(p, F, L, current_label) /* procedure for detection
   and labeling of plateau */
2: Input: p, F, L, current_label.
```

```
3: Output: L, current_label.
4: L(p) ← current_label;
5: FIFO_INIT(Q_M);  /* initialize the FIFO Queue (Q_M) */
6: FIFO_ADD(p, Q_M);  /* insert the pixel p into the queue Q_M */
7: PA_Flag ← FALSE;  /* initialize the plateau analysis flag */
8: while fifo queue Q_M not empty {/* plateau Detection phase */} do
9:    p ← FIFO_REMOVE(Q_M);  /* dequeue the pixel p from the queue Q_M */
10:   for all q ∈ N_G^8(p) {/* N_G^8 p): 8-connected neighbors of p */} do
11:      if ((L(q) = INIT) and (F(p) = F(q))) then
12:         L(q) ← current_label;
13:         FIFO_ADD(q, Q_M);  /* insert the pixel q into the queue Q_M */
14:      else if (F(q) < F(p)) then
15:         PA_Flag ← TRUE;
16:      end if
17:   end for
18: end while
19: if PA_Flag = TRUE {a non-minimum plateau is detected} then
20:   FIFO_ADD(p, Q_M); {/* insert the pixel q into the queue Q_M */}
21:   while fifo queue Q_M not empty {/* plateau Detection phase */} do
22:      p ← FIFO_REMOVE(Q_M);  /* dequeue the pixel p from the queue Q_M */
23:      for all q ∈ N_G^-(p) {/* N_G^-(p): neighbors of p during reverse scan */} do
24:         if (L(q) = current_label ) then
25:            L(q) ← NARM;
26:            FIFO_ADD(q, Q_M);
27:         end if
28:      end for
29:   end while
30: else if PA_Flag = FALSE {a minimum plateau is detected} then
31:   current_label := current_label + 1;
32: end if
33: /* End of procedure Plateau_Analysis_Labeling */
```

3.2.2 Simulated Flooding

At this stage, one has detected and labelled local minima and placed them in an array S. For each local minimum, its 8-connected neighbors are inspected. Initialize the queue Q_F with the local minima (from array S) which has at least one non-labeled neighborhood pixel. Pixel p is dequeued from the queue Q_F one at a time and its label is assigned to the non-labelled neighboring pixel q ($q \in N_G(p)$, where $N_G(p)$ may be either 8-connected neighbors or 4-connected neighbors of p denoted by $N_G^8(p)$ and $N_G^4(p)$) by using conditional comparisons among the neighbors of p of higher altitude. For each non-labelled neighboring pixel q with gray value greater than or equal to that of p, its 4-connected neighbors $N_G^4(q)$ (which are also neighbors of p) and its 8-connected supporting pixels $N_G^8(q)$ are compared with the center pixel p. If the flag $Flag_1$ is TRUE, then the 8-connected supporting pixels would be compared with the center pixel p. If the flag $Flag_2$ is TRUE, then the pixel q would derive its label from the center pixel p. Next q is inserted into the queue Q_F. Flooding process stops when all the elements in the array S have been visited. The pseudo-code of the simulated flooding process is as follows.

Procedure 3.4 *Simulated_Flooding(F, L) /* procedure for simulated flooding */*
Input: Morphological multi-scale gradient image F $\{F : D_F \rightarrow \mathbb{N}\}$
Output: Labelled image L $\{L : D_f \rightarrow \mathbb{N}\}$
Let G be a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$ and $N_G(p)$ stands for the neighbors of a pixel p on the grid G.
Store the labelled local minima in an array S of size M.
for all m $\in$ M {/* visit all the local minima which have at least one non-labelled neighbor */} **do**
  FIFO_INIT(Q_F);
  FIFO_ADD($S(m)$, Q_F) /* initialize the queue with the present local minima */
  while fifo queue Q_F not empty {/* find the catchment basin around the local minima */} **do**
    $p \leftarrow$ FIFO_REMOVE(Q_F);
    for all (q $\in N_G(p)$ with $L(q) = NARM$) {/* for all neighbors of p */} **do**
      $Flag_1 \leftarrow$ TRUE; $Flag_2 \leftarrow$ TRUE;
      Four_Conn_Neigh_Comp(F, p, q, $Flag_1$); /* procedure for Four connected neighbor hood comparison */
      if ($Flag_1$ = TRUE) **then**
        Eight_Conn_Sup_Neigh_Comp(F, p, q, $Flag_2$); /* procedure for Eight connected supporting neighborhood comparison */
        if ($Flag_2$ = TRUE) **then**

17: $L(q) \leftarrow L(p)$;
18: FIFO_ADD(q, Q_F)
19: **end if**
20: **end if**
21: **end for**
22: **end while**
23: **end for**
24: /* End of procedure Simulated_Flooding */

Procedure Four_Conn_Neigh_Comp used in the procedure Simulated_Flooding may be described by means of pseudo-code as follows:-

---

1: **Procedure 3.4.1** *Four_Conn_Neigh_Comp(F, p, q, $Flag_1$) /* procedure for 4-connected neighborhood $N_G^4(q)$ comparison */*
2: Input: F, p, q and $Flag_1$.
3: Output: $Flag_1$.
4: **for all** r $\in$ (N_G^4(q) $\wedge$ N_G(p)) {/* compare 4-connected neighbors of q */} **do**
5: **if** ($F(r) < F(p)$) **then**
6: $Flag_1 \leftarrow$ FALSE;
7: **end if**
8: **end for**
9: /* End of procedure Four_Conn_Neigh_Comp */

Next, procedure Eight_Conn_Sup_Neigh_Comp used in the procedure Simulated_Flooding may be given in pseudo-code form as follows:-

---

1: **Procedure 3.4.2** *Eight_Conn_Sup_Neigh_Comp(F, p, q, $Flag_2$) /* procedure for 8-connected neighborhood $N_G^8(q)$ comparison */*
2: Input: F, p, q, $Flag_2$
3: Output: $Flag_2$
4: **for all** r $\in$ (N_G(q) $-$ ((N_G(q) $\wedge$ N_G(p))$\vee$ $\{p\}$)) {/* compare 8$-$connected neighbors of q which are not neighbors of p */} **do**
5: **if** ($F(r) < F(p)$) **then**
6: $Flag_2 \leftarrow$ FALSE;

7: **end if**

8: **end for**

9: /* End of procedure Eight_Conn_Sup_Neigh_Comp */

3.2.3 Complexity Analysis

This section analyzes the complexity of the major procedures of the proposed algorithm. Let $n = M \times N$ be the dimension of the image F whose domain is denoted as $D_F \subset \mathbb{Z}^2$, and G is a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$. $N_G(p)$ stands for the neighbors of a pixel p on the grid G.

Local minima detection and labelling

Let p_1, p_2, p_3, ..., p_P be P plateaus, each p_i having n_i pixels and P_N be the detected non-minima plateaus.

1. The number of comparisons for plateau detection and plateau analysis (lines 10-17, procedure 3.3.2) is $\sum_{i=1}^{P} n_i * N_G^8$, where N_G^8 : 8 - connected neighborhood pixels.

2. The number of comparisons for non-minima plateau labeling (lines 19-32, procedure 3.3.2) is $\sum_{i=1}^{P_N} n_i * N_G^-$, where N_G^- : Backward raster scan neighborhood pixels.

Total number of comparisons for local minima detection and labeling (on an average case) is then

$$LMD = \sum_{i=1}^{P} n_i * N_G^8 + \sum_{i=1}^{P_N} n_i * N_G^- \simeq 2n \text{ (two raster scans of the image)} \approx O(n) \qquad (3.2.3)$$

Simulated flooding

Let l_1, l_2, l_3, ..., l_{LM} be the plateaus of local minima in the image F; moreover, let l_j have m_j pixels which have at least one NARM neighborhood pixel. Let C be the present center pixel and N_1, N_2, ..., N_8 the 8-connected neighbors of the center pixel. The number of comparisons for deciding the labels of the neighboring pixels (N_1, N_2, ..., N_8) of the center pixel C is determined by performing conditional comparisons.

Stage 1: To compare the intensity value of the center pixel C with those of its immediate 8-connected neighbors $N_G^8(C)$, the number of comparisons required is $N_G^8(C)$.

Stage 2: To compare the center pixel with the $4-$ connected neighbors N_1, N_2, ..., N_8, which are also neighbors of the center pixel C (procedure 3.4.1), $\sum_{i=1}^{8}(N_G^4(N_i) \wedge N_G^8(C))$,
where $N_G^4(N_i)$: $4-$connected neighbors of N_i.

Stage 3: To compare the center pixel with its $8-$ connected neighbors N_1, N_2, ..., N_8, which are not neighbors of the center pixel C (procedure 3.4.2), $\sum_{i=1}^{8}(N_G^8(N_i) - ((N_G^8(N_i) \wedge N_G^8(C)) \vee \{C\})$,
where $N_G^8(N_i)$: $8-$connected neighbors of N_i.

The number of comparisons for deciding the labels of the neighboring pixels of center pixel C, hence given by the following expression namely

$$NLP = N_G^8(C) + \sum_{i=1}^{8}(N_G^4(N_i) \wedge N_G^8(C)) + \sum_{i=1}^{8}(N_G^8(N_i) - ((N_G^8(N_i) \wedge N_G^8(C)) \vee \{C\})) \quad (3.2.4)$$

Thus the total number of comparisons for the entire label propagation (on an average case) is

$$LP = n + \frac{1}{P_N}(\sum_{i=1}^{P_N} n_i) * (\sum_{j=1}^{L} m_j * NLP) \simeq 2n \quad (3.2.5)$$

Total number of comparisons for the entire watershed computation (on an average case) is

$$LMD + LP \approx 4n \text{ (four raster scans of the image)} \approx O(n) \quad (3.2.6)$$

3.3 Simulation Results

The proposed algorithm has been implemented under Linux 9.0 environment using POSIX Threads on a Pentium IV 1.5 GHz computer and applied on various test images to verify its effectiveness. Tables 3.1 and 3.2 show that the proposed algorithm is faster than Meyer's algorithm. The number of comparisons for label propagation while running both the algorithms for different test images is plotted in Figure 3.4. Also, the segmentation results are obtained for different values of the threshold h involved in the multi-scale gradient operation [Wan97]. It may be mentioned here that the parameter h used for reducing over-segmentation needs to be specified manually [Wan97, Wan98] as its appropriate value depends on the nature of variation of gray values in the image at hand. Figure 3.5 shows that an increase in h leads to a reduction in the number of regions. Variation of simulation time with the values of h is plotted in Figure 3.6. Results of running Meyer's algorithm and the proposed algorithm on a multi-scale morphological gradient [Wan97] version of the Tennis image, the Cermet image, the MRI Brain image, the Washington_ir image, the Steel fissure image and the Blood cell image appear in Figures 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12 respectively.

Test Image	Threshold value(h)	Number of homogeneous regions	Time (Sec)	
			Meyer's algorithm	Proposed algorithm
Claire (352×288)	10	17	0.125	0.11
Tennis (352×288)	13	22	0.123	0.107
Heart (256×256)	8	14	0.093	0.086
Akiyo (164×144)	14	22	0.036	0.039
MRI Brain (256×256)	12	38	0.092	0.084
Cermet (256 × 256)	30	61	0.096	0.082
Washinton_IR (250×250)	16	46	0.116	0.078
Steel fissure (290×369)	15	10	0.125	0.102
Blood Cells (212×353)	26	20	0.091	0.074

Table 3.1: Comparison between simulation time (in seconds) of Meyer's algorithm and the proposed algorithm.

Test Image	Threshold value (h)	Number of regions	No. of Comparisons for	
			Meyer's label propagation	Proposed label propagation
Claire (352×288)	10	17	1,59,568	1,48,345
Tennis (352×288)	13	22	2,85,925	2,75,094
Heart (256×256)	8	14	2,27,692	2,20,113
Akiyo (164×144)	14	22	1,05,638	1,02,640
MRI Brain (256×256)	12	38	1,89,580	1,88,218
Cermet (256 × 256)	30	61	3,23,168	3,02,395
Washinton_IR (250×250)	16	46	3,92,790	3,80,841
Steel Fissure (290×369)	15	10	3,34,864	2,87,461
Blood Cells (212×353)	26	20	1,52,383	1,17, 455

Table 3.2: Number of comparisons for simulated flooding (label propagation).

3.3.1 Performance Analysis

Following a region-based performance evaluation scheme [HD95], the segmentation quality of the proposed algorithm is quantitatively assessed in terms of size and location of the segmented regions. Let S_1 and S_2 be the segmented images produced by Meyer's original algorithm and the proposed algorithm respectively, and expressed as

$$S_1 = \{ R_1^1, R_1^2, R_1^3, \ldots, R_1^n \}$$

$$S_2 = \{ R_2^1, R_2^2, R_2^3, \ldots, R_2^n \}$$

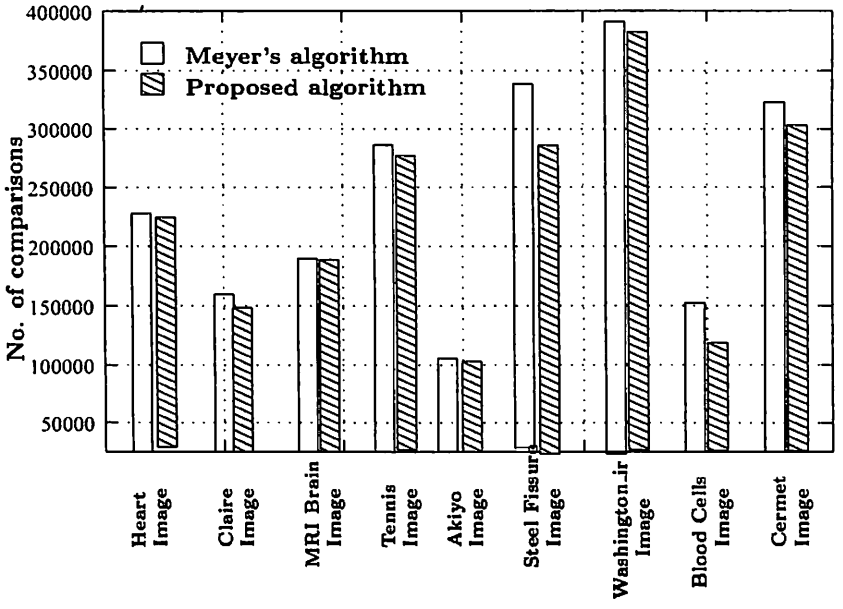


Figure 3.4: Number of comparisons for label propagation while running Meyer's algorithm and proposed algorithm for different test images.

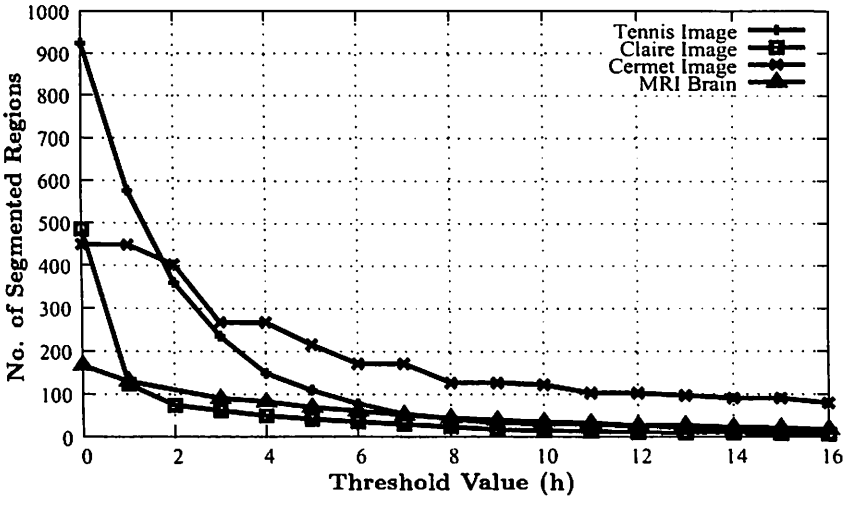


Figure 3.5: Number of regions for different threshold value (h).

The directional Hamming distance from S_1 to S_2 is defined as

$$D_H(S_1 \Rightarrow S_2) = \sum_{R_2^i} \sum_{R_1^k \neq R_1^j, R_1^k \cap R_2^i \neq \emptyset} | R_2^i \cap R_1^k |, \tag{3.3.1}$$

where $|.|$ denotes the size of a set. Therefore, $D_H(S_1 \Rightarrow S_2)$ is the total area under the non-maximal intersections between all regions $R_2^i \in S_2$ and all regions $R_1^k \in S_1$. The reverse distance $D_H(S_2 \Rightarrow S_1)$ can also be similarly computed. The region-based performance measure based on normalized Hamming

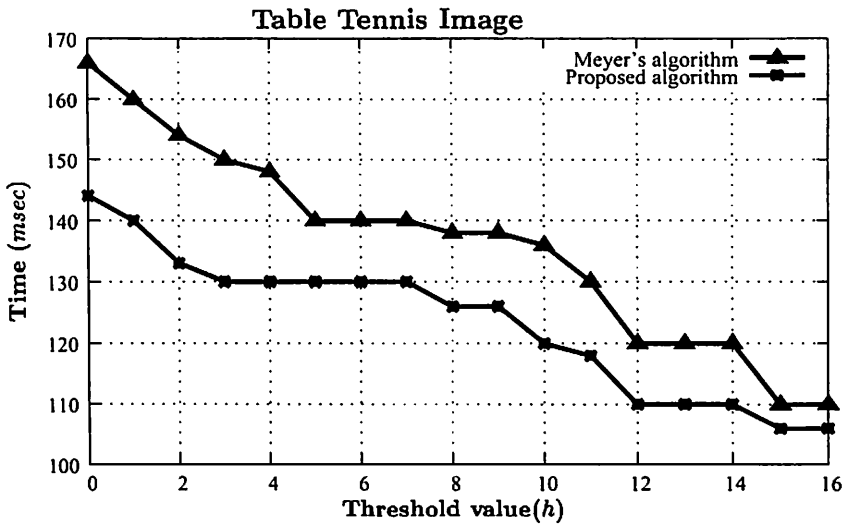


Figure 3.6: Simulation time while running Meyer's and proposed algorithms on Tennis image.

distance is given as

$$P = 1 - \frac{D_H(S_1 \Rightarrow S_2) + D_H(S_2 \Rightarrow S_1)}{2\times \mid S \mid}, \tag{3.3.2}$$

where $\mid S \mid$ is the image size and $P \in [0,1]$. The smaller the degree of mismatch, the closer the measure P is to one. The goal is to quantitatively describe the mismatch between S_1 and S_2. The performance measure of the proposed algorithm evaluated for various images is provided in Table 3.3, which shows that the homogeneous regions are identified by Meyer's algorithm and the proposed watershed algorithm in an almost identical manner ($P \approx 1$).

Test Image	Threshold value (h)	Number of homogeneous regions	mismatched pixels	Performance measure (P)
Claire (352×288)	10	17	539	0.994688
Tennis (352×288)	13	22	627	0.992661
Heart (256×256)	8	14	978	0.986481
Akiyo (164×144)	14	22	399	0.982782
MRI Brain (256×256)	12	38	1023	0.981812
Cermet (256 × 256)	30	61	714	0.985474
Washinton_IR (250×250)	16	46	1205	0.975856
Steel Fissure (290×369)	15	10	264	0.997523
Blood Cells (212×353)	26	20	572	0.992357

Table 3.3: Quantitative evaluation of the segmented images

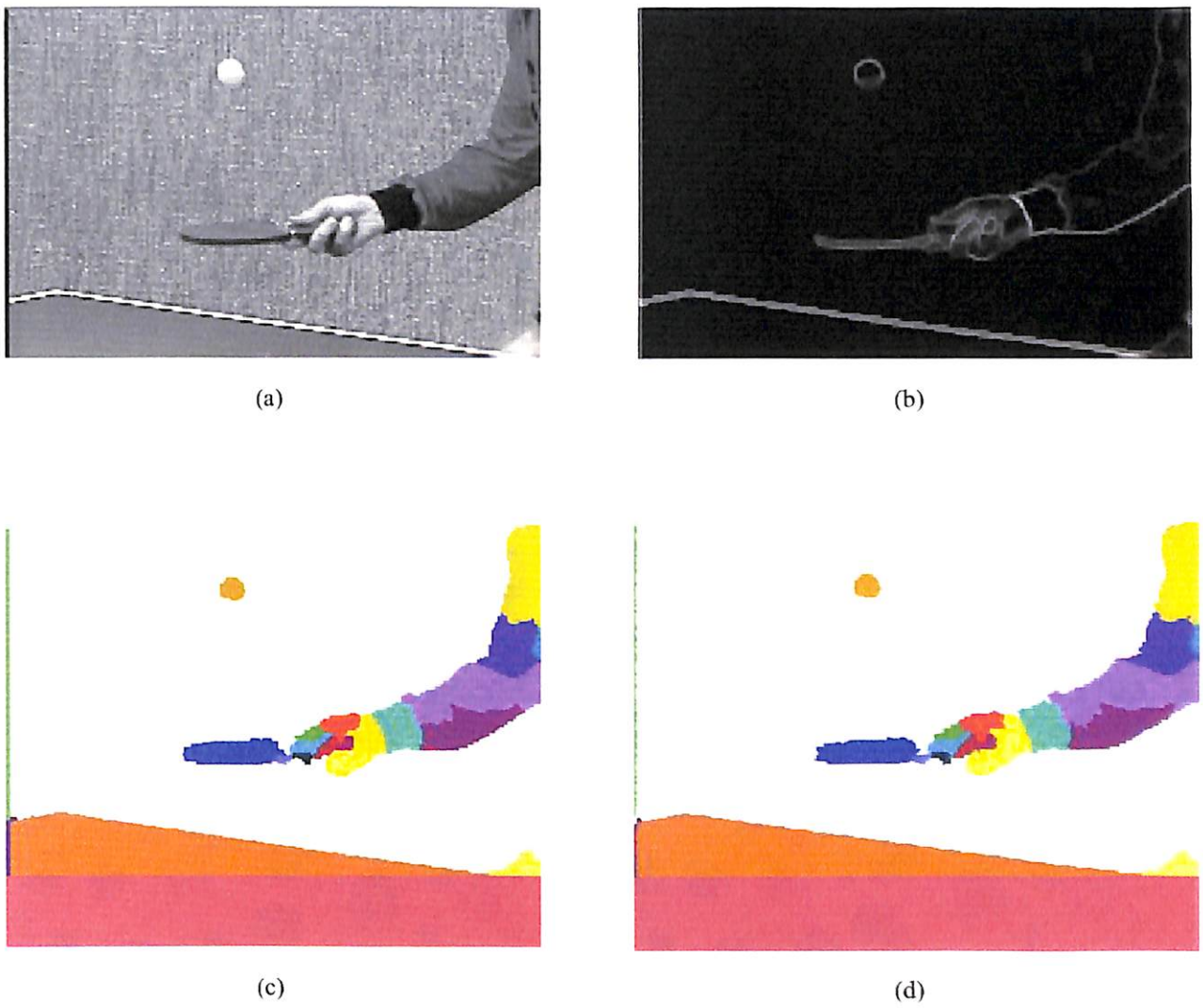


Figure 3.7: (a) Original Tennis image; (b) Morphological multi-scale gradient operation on Table Tennis image with a threshold value of 13; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

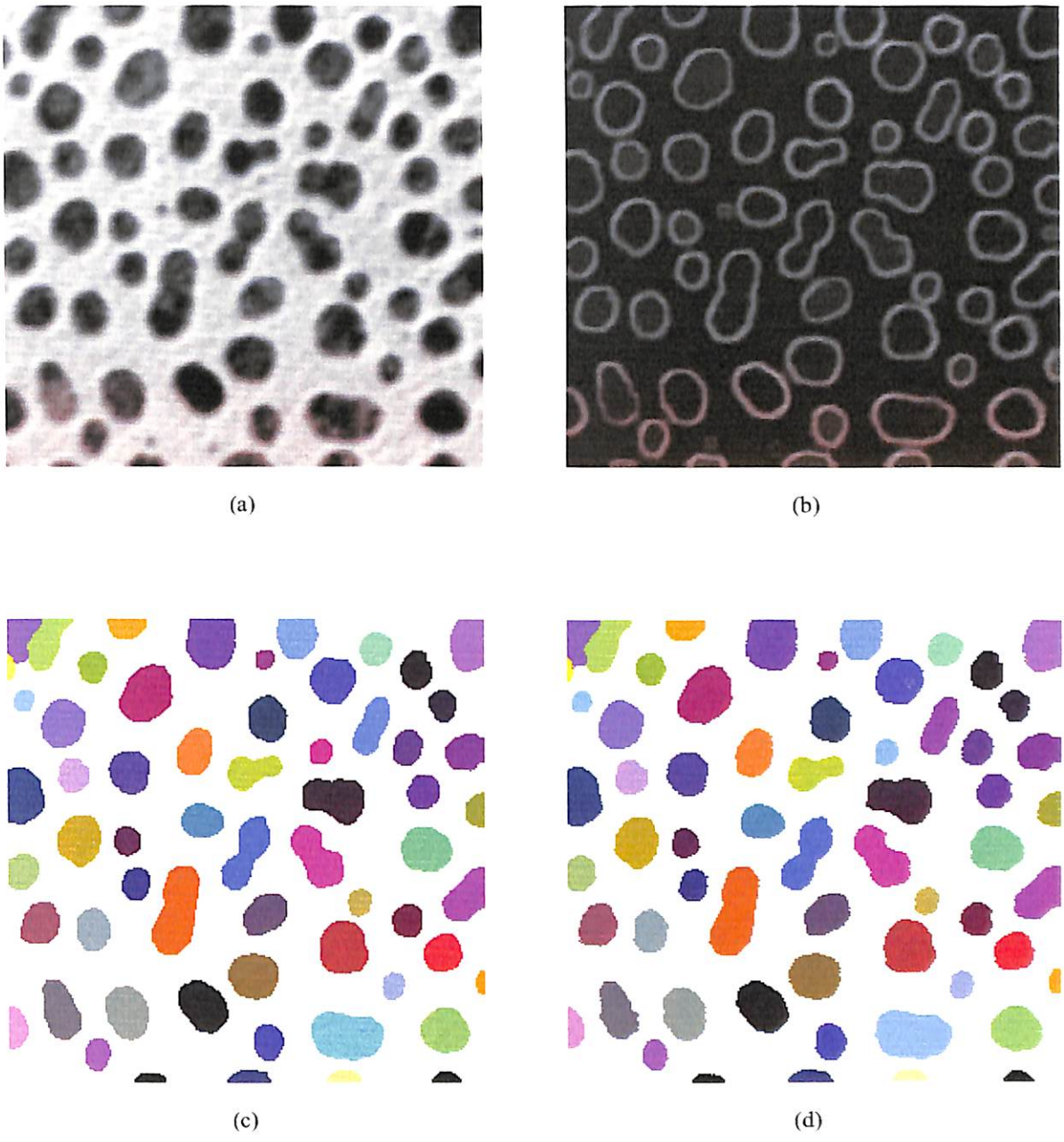


Figure 3.8: (a) Original Cermet image; (b) Morphological multi-scale gradient operation on Cermet image with a threshold value of 30; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

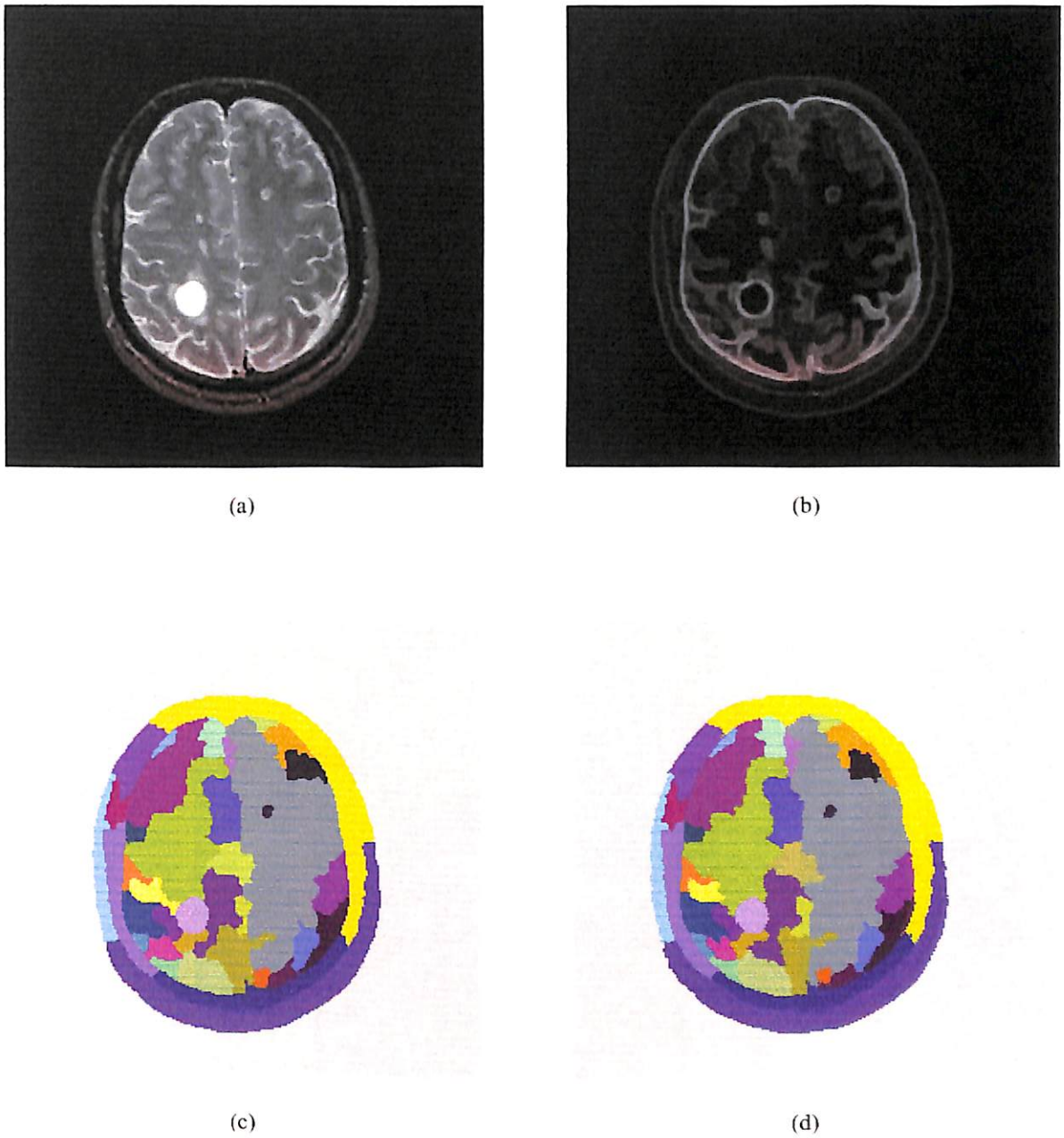


Figure 3.9: (a) Original MRI Brain image; (b) Morphological multi-scale gradient operation on MRI Brain image with a threshold value of 12; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

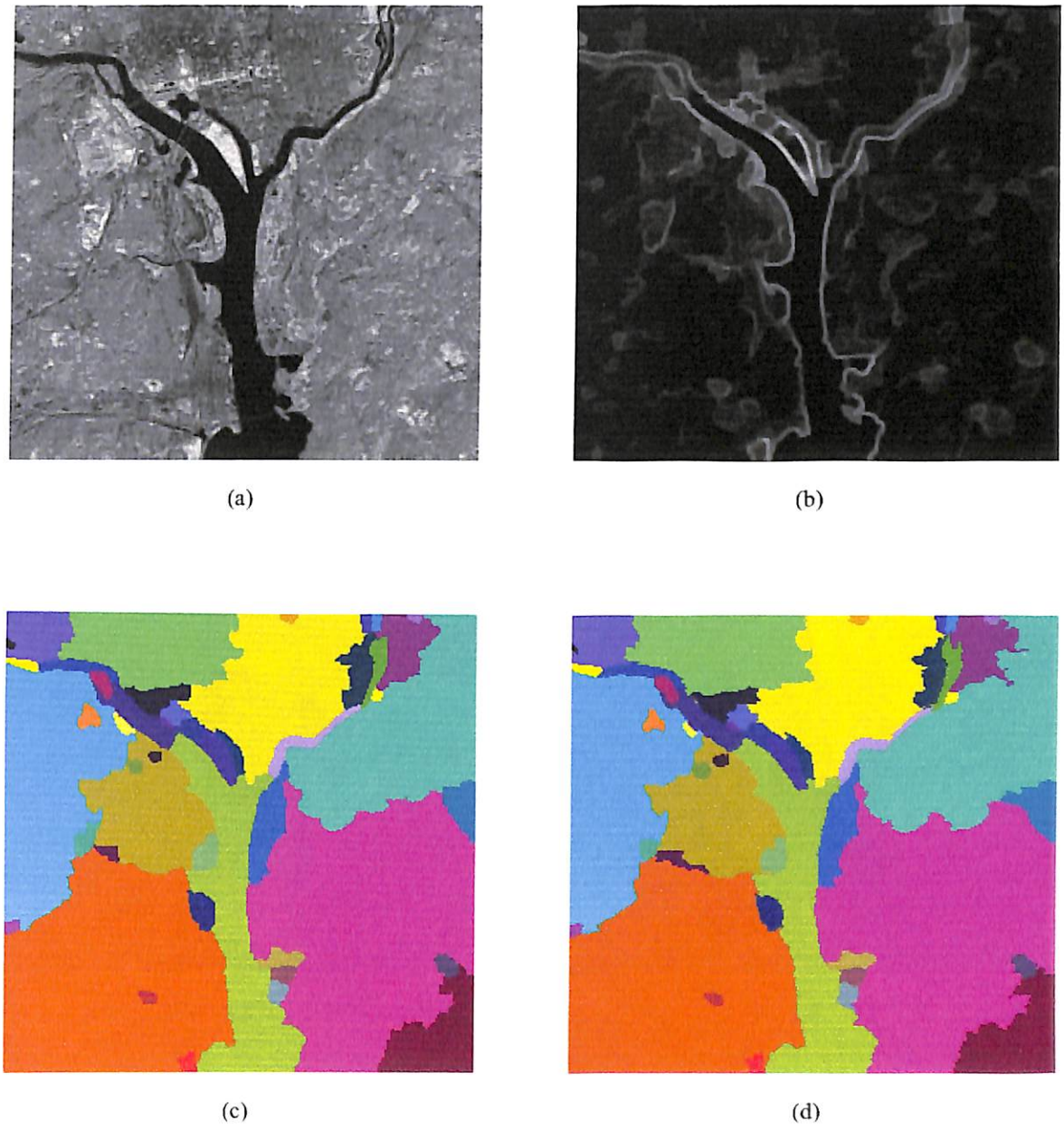


Figure 3.10: (a) Original Washington_ir image; (b) Morphological multi-scale gradient operation on Washington_ir image with a threshold value of 16; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

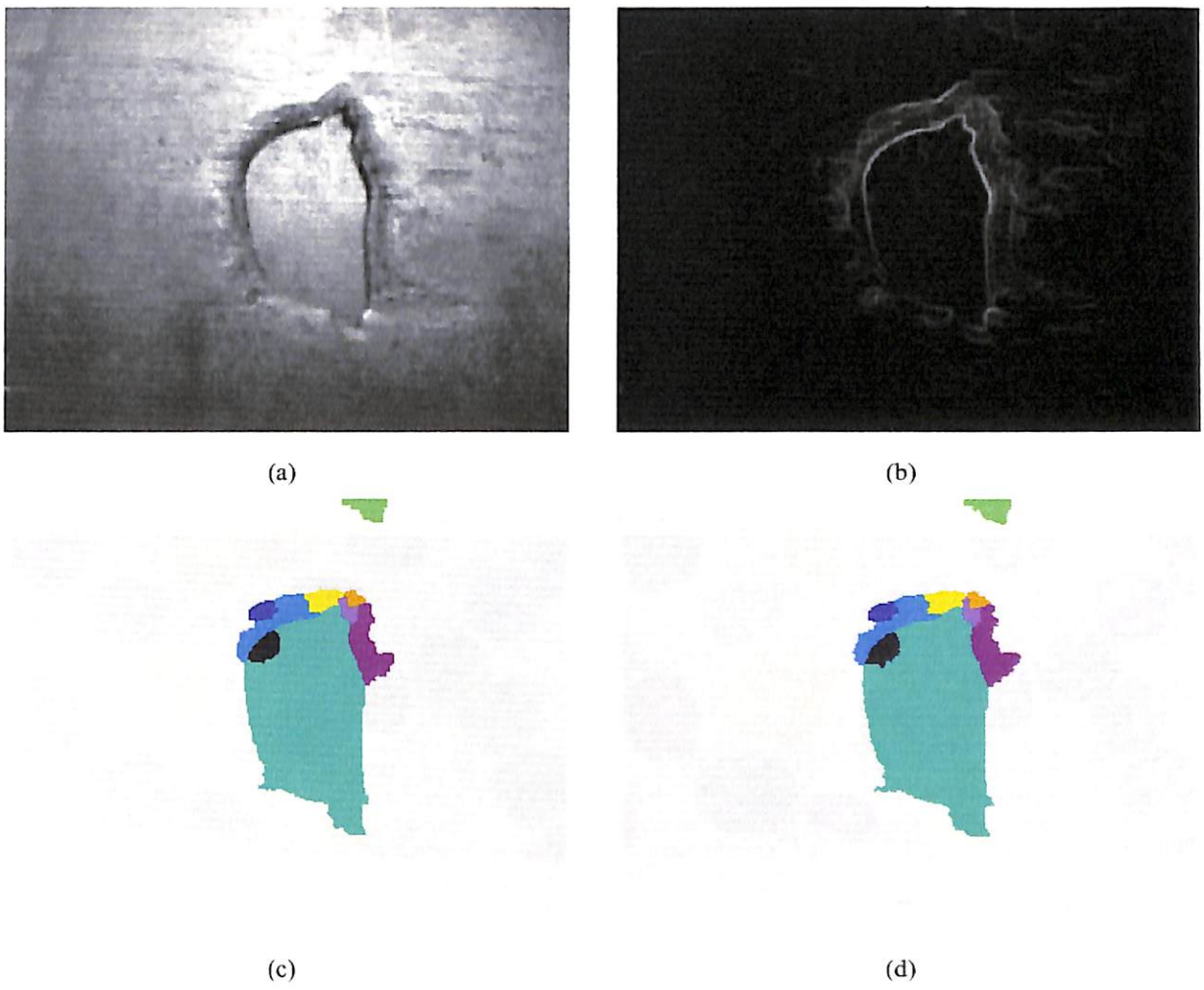


Figure 3.11: (a) Original Steel Fissure image; (b) Morphological multi-scale gradient operation on Steel Fissure image with a threshold value of 15; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

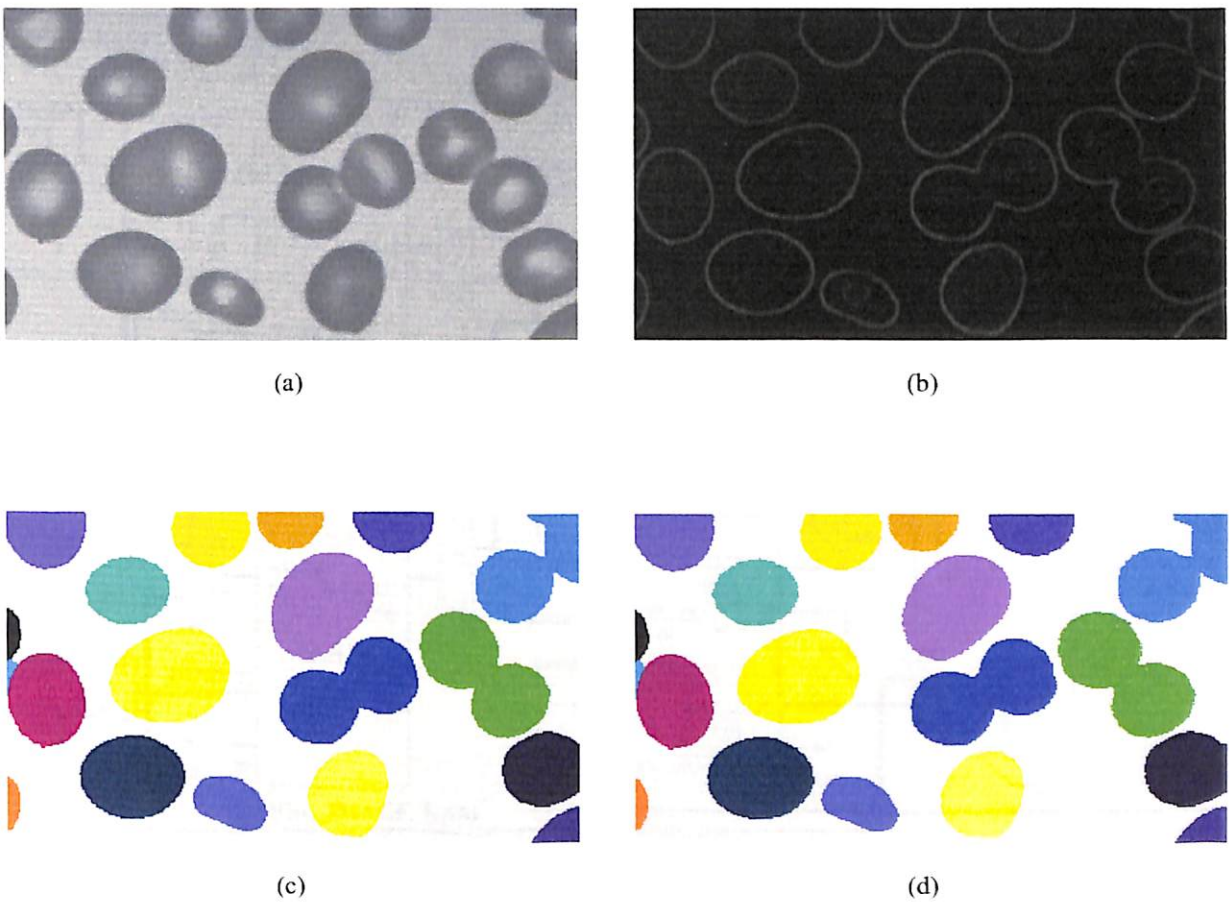


Figure 3.12: (a) Original Blood cell image; (b) Morphological multi-scale gradient operation on Blood cell image with a threshold value of 26; (c) Segmentation produced by Meyer's watershed algorithm; (d) Segmentation produced by the proposed watershed algorithm.

3.4 Proposed Prototype Architecture and Implementation

The present section describes the hardware architecture that implements the improved watershed algorithm proposed in the previous section. The block diagram of the overall architecture that implements the proposed watershed algorithm is shown in Figure 3.13. This architecture consists of three mem-

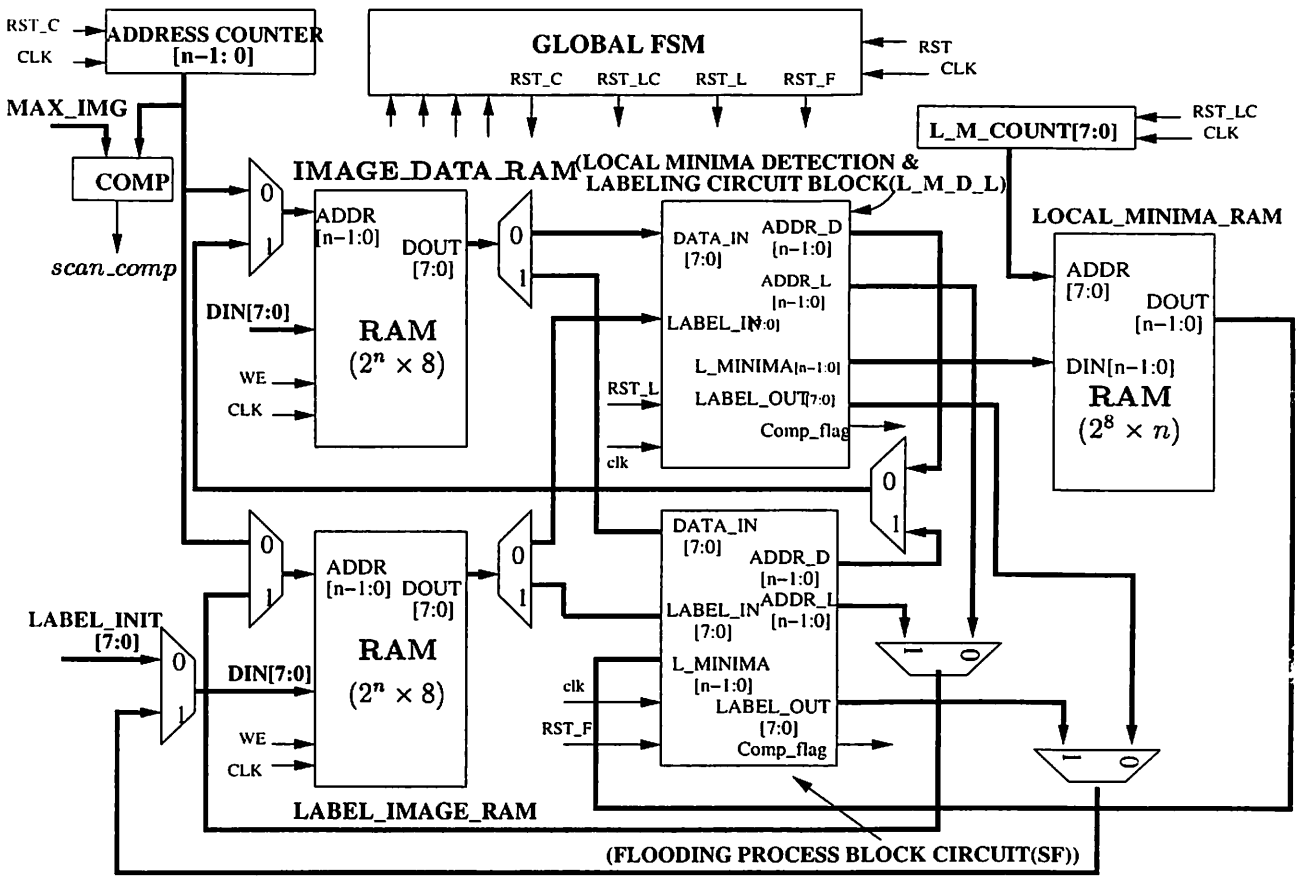


Figure 3.13: Complete architecture of the watershed algorithm realization

ory modules, which are meant to hold the input image (IMAGE_DATA_RAM), the output image of labels (LABEL_IMAGE_RAM) and the local minima data (LOCAL_MINIMA_RAM), and two major hardware blocks meant for local minima detection and labeling (L_M_D_L), and simulated flooding (SF) process. In addition, it consists of a control unit (FSM) which generates the signals required to control the MUXes, DEMUXes, read/write signals for the RAMs and set/reset signals for ADDRESS COUNTER, local minima detection and labelling block, and simulated flooding block. Raster scan can be performed by using the memory address counter (ADDRESS COUNTER). A control signal *scan_complete* is generated, when the memory address counter reaches the last address of the image. In

addition, the L_M_COUNTER is used to select the local minima data in the LOCAL_MINIMA_RAM. A brief description of the individual hardware blocks follows.

3.4.1 Local Minima Labeling Circuit

The pipelined architecture for performing detection and labeling of local minima is shown in Figure 3.14. This circuit consists of four functional blocks namely, 8−connected neighborhood address generation block (NAG), plateau analysis block (PA), plateau detection block (PD) and plateau labelling block (PL). The NAG block generates the neighborhood addresses (N_ADDR_1 − N_ADDR_8) of the center pixel (B_ADDR) and load the registers with the corresponding pixel data (N_DATA_1 − N_DATA_8) and label values (N_LABEL_1 − N_LABEL_8) from IMAGE_DATA_RAM and LABEL_IMAGE_RAM respectively. The PA block compares the center pixel value with the neighboring pixels and decides the label of the plateau. The PD block detects the forward raster scan neighbors that have the same pixel value as that of the center pixel. The detected pixels are marked as visited (−2) and enqueued in the FIFO QUEUE which can be further explored. The PL block labels the entire plateau which has been decided by PD. This circuit detects the reverse raster scan neighbors which have the same label −2 and are marked as "pixels with label decided".

The controller LMD_FSM generates the set/reset signal for NAG, PA, PD and PL. Moreover, it generates the signals to control MUXes, DEMUXes, read/write for FIFO, counter enable signal for the address counter and the label counter L_M_COUNT. The control sequence of the above circuit is as follows.

Step 1: Reset the address generation counter (ADDRESS COUNTER), label counter (LABEL COUNTER) and the counter L_M_COUNT. Initialize all the flags to zero.

Step 2: Read data in DATA_IN from IMAGE_DATA_RAM addressed by ADDRESS COUNTER. Read label in LABEL_IN from LABEL_IMAGE_ RAM addressed by ADDRESS COUNTER.

Step 3: Compare LABEL_IN with -1;

if L_C_Flag is not 0 **then** set the reset signals RST_NAG, RST_PA, RST_PD, RST_PL to 1;

else set F_RST to 0.

Write the visited Flag (-2) in LABEL_IMAGE_RAM pointed by ADDRESS COUNTER

Goto Step 4.

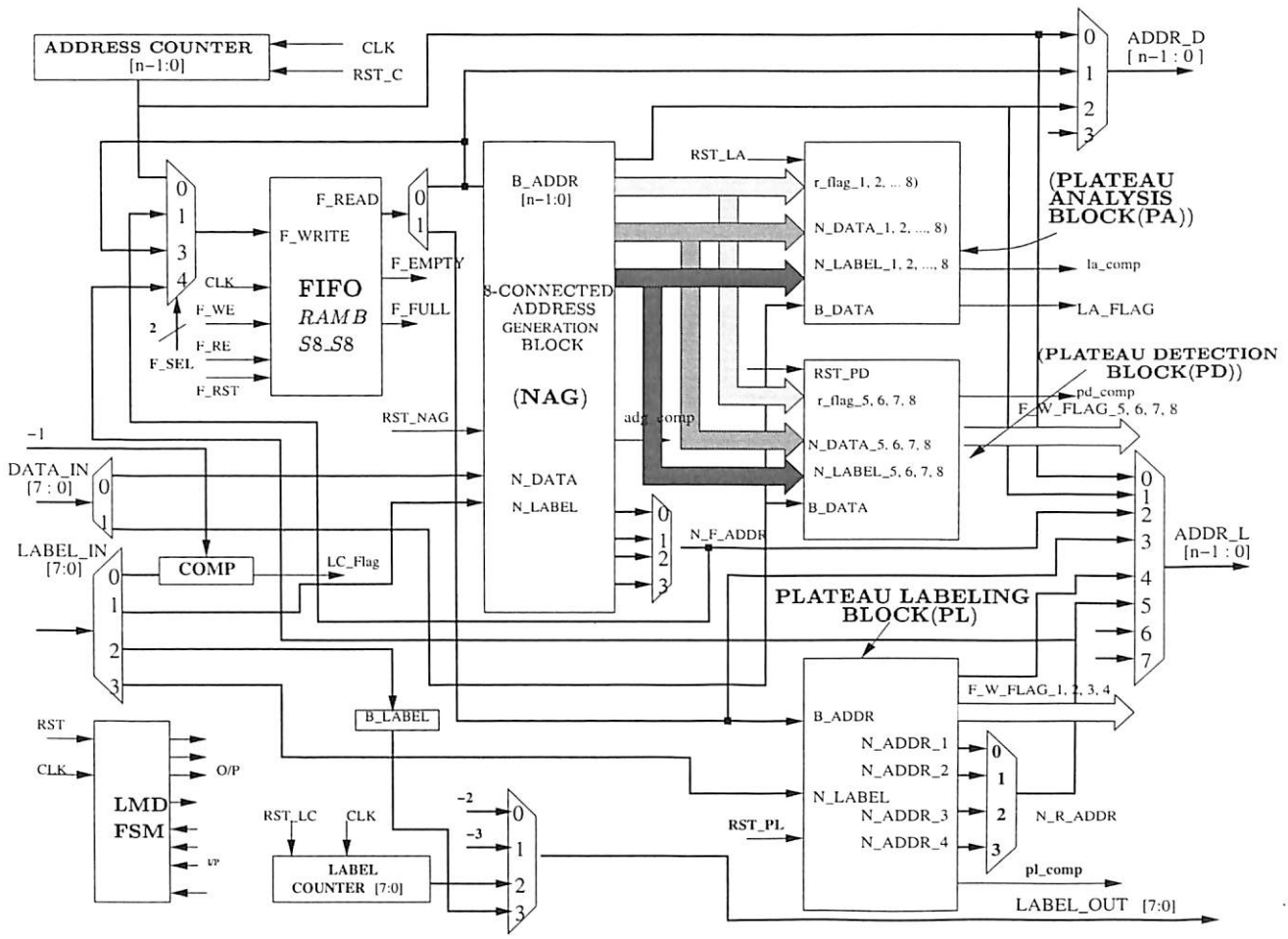


Figure 3.14: Circuitry for local minima detection and labeling

Step 4: Increment the ADDRESS COUNTER.

If ADDRESS COUNTER value is equal to the maximum image size **then** Goto Step 11; **else** Step 2.

Step 5: Set RST_NAG to 0.

If the address generation completion signal adg_comp is 1 **then** set the reset signals RST_PA, RST_PD TO 0.

Step 6: **If** pd_comp is 1 **then**

write the visited Flag (-2) in LABEL_IMAGE_RAM pointed by N_F_ADDR;

Enqueue N_F_ADDR in FIFO.

Step 7: **If** F_EMPTY is 1 **then** set F_RST to 1.

else dequeue the value (B_ADDR) from FIFO. Goto Step 5.

Step 8: **If** PA_Flag is 1 **then**

Write the N_R_M (-3) in LABEL_IMAGE_RAM pointed by F_READ.

else write the present label counter value in LABEL_IMAGE_RAM pointed by F_READ.

Write the ADDRESS COUNTER value in LOCAL_MINIMA_RAM pointed by L_M_COUNT.

Increment the label counter. Increment the L_M_COUNT.

Step 9 Set F_RST, RST_PL TO 0.

If pl_comp is 1 **then**

Write the B_LABEL in LABEL_IMAGE_RAM pointed by N_R_ADDR.

Enqueue N_R_ADDR in FIFO.

Step 10 **If** F_EMPTY is 1 **then** set F_RST to 1. Goto Step 4.

else dequeue the value (B_ADDR) from FIFO. Goto Step 9.

Step 11 Exit. /* End of the local minima detection and labeling */

The functional principal modules of the local minima detection and labeling architecture are described as follows.

Memory Block

RAMs are required to store images. In our case, four memory blocks are required to store the Input Gradient Image, the Label Output Image, the Distance Image and the Steepest Lower Image. Block diagram of RAMs is shown in Figure 3.15.

Two types of RAM namely single-port RAM and dual-port RAM have been used. In a dual-port RAM, both read and write operations on different address can be performed in a single cycle. Many times we need to check and assign label/distance to a pixel of Label/Distance Images. The dual-port RAMs have been used for Label and Distance Images, and single-port for other Images.

The size of RAMs block can be configured as follows.

Address lines are, $N = \lceil \log_2(m \times n) \rceil$bits, where $m \times n$ is image size.

Data lines are 8-bits wide.

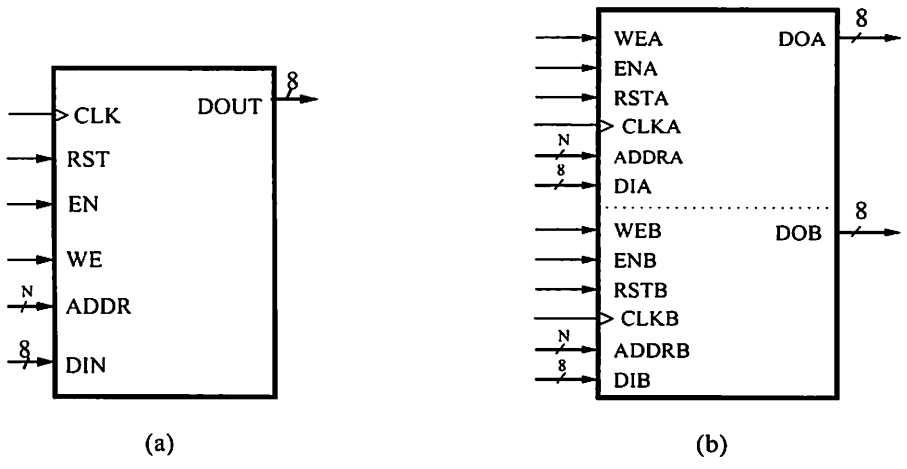


Figure 3.15: RAM blocks. (a) Single-port RAM. (b) Dual-port RAM.

FIFO Queue

The schematic diagram of the synchronous FIFO is shown in Figure 3.16. This can be designed by using a Dual-Port RAM. Two binary counters are used as the read and write address counters. To perform a read, the read enable signal (read_enable) is driven high prior to a rising clock edge. The read data signal (read_data) will be presented on the output during the next clock cycle. To do a burst read, simply leave read_enable high for as many clock cycles as are required. The last word has been read when Empty becomes active after a read. If another read is attempted, read_data will be invalid.

To preform a write, the write data (write_data) must be present on the inputs, and the write enable (write_enable) signal must be driven prior to the rising clock edge. As long as the flag FULL is not set, data will be written into the FIFO. To do a burst write, the write_enable is left high, and a new write_data must be available every cycle.

A FIFO counter (fifo_count) is added for convenience, to determine the FIFO contents. It is a binary counter of the number of words currently stored in the FIFO. The flag EMPTY is set when the fifo_counter state (value) zero, or when the fifo_count is one and only a Read is being performed. Similarly, the Full flag is set when the fifo_count is $2^N - 1$(upper limit), or when the fifo_count is $2^N - 2$ and only a write is being performed. During global reset (fifo_gsr), both these signals are driven High, to prevent any external logic from interfacing with the FIFO during this time. The performance of the 255×8 FIFO as implemented on a XILINX XCV400HQ240 device is limited to 139.76MHz. It utilizes 1% of CLB slices.

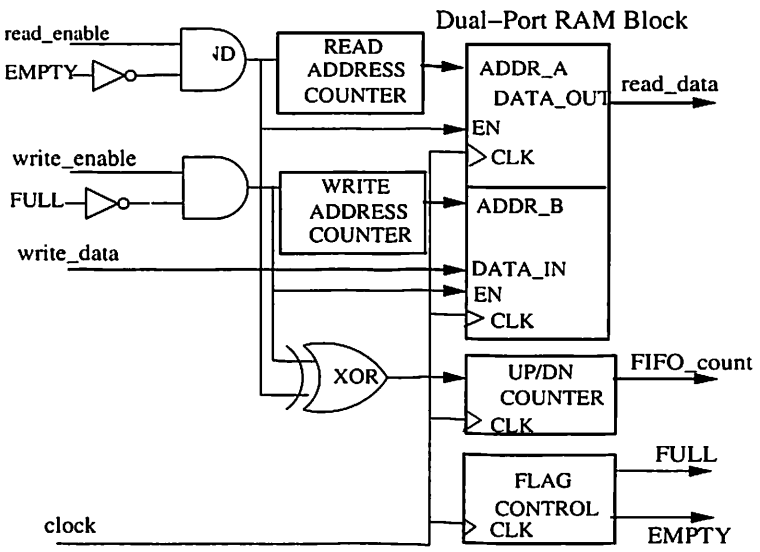


Figure 3.16: The schematic diagram of FIFO implementation

Neighborhood Address Generation (NAG) Block

The neighborhood addresses are generated in parallel through addition/subtraction of the Image_Width and the B_address as shown in Figure 3.17(a). The schematic diagram of N_1 address generation module is shown in Figure 3.17(b). The circuit blocks labeled ADD and 2's COMP_SUB are used to generate the neighbor pixel addresses from the center pixel address. The block REM is a remainder circuit which is used to find the validity of the generated neighbor address. If the center pixel belongs to a boundary, some neighbor addresses may be incorrectly computed. To prevent this, the remainder left on dividing by the B_Address and the Image_Width is computed. The remainder determines the validity of the neighboring pixel (whether it is inside or outside the image) as the read_flag is generated.

Plateau Detection Block (PD)

The plateau detection is performed by comparing the center pixel data (B_DATA) with each of its 8 nearest neighbors (N_DATA_1, ..., N_DATA_8) in parallel, as depicted in Figure 3.18(a). The schematic diagram of the PD_N (plateau detection for single neighbor) block is shown in Figure 3.18(b). The f_w_flag signal is generated when the neighbor pixel data (N_DATA) and the label image data (N_LABEL) are equal to the center pixel data (B_DATA) and the initialized value of the label image INIT respectively. It indicates that the neighboring pixel belongs to its plateau and not visited previously. This control signal invokes the control unit, so that the control unit can generate the necessary

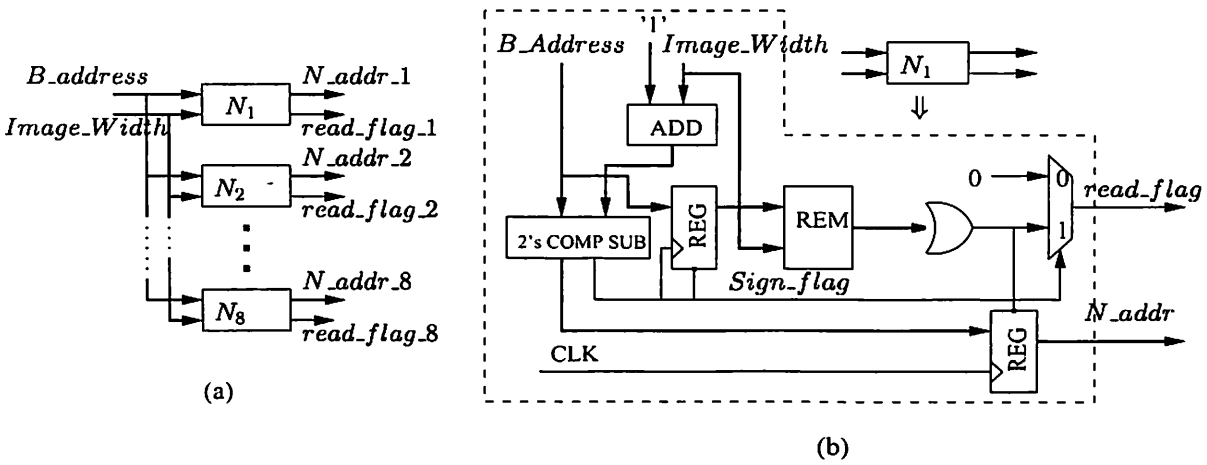


Figure 3.17: (a). The block diagram of the NAG Block. (b). Schematic diagram for N_1 address generation block.

signal to enqueue the address of the neighbor pixel into FIFO.

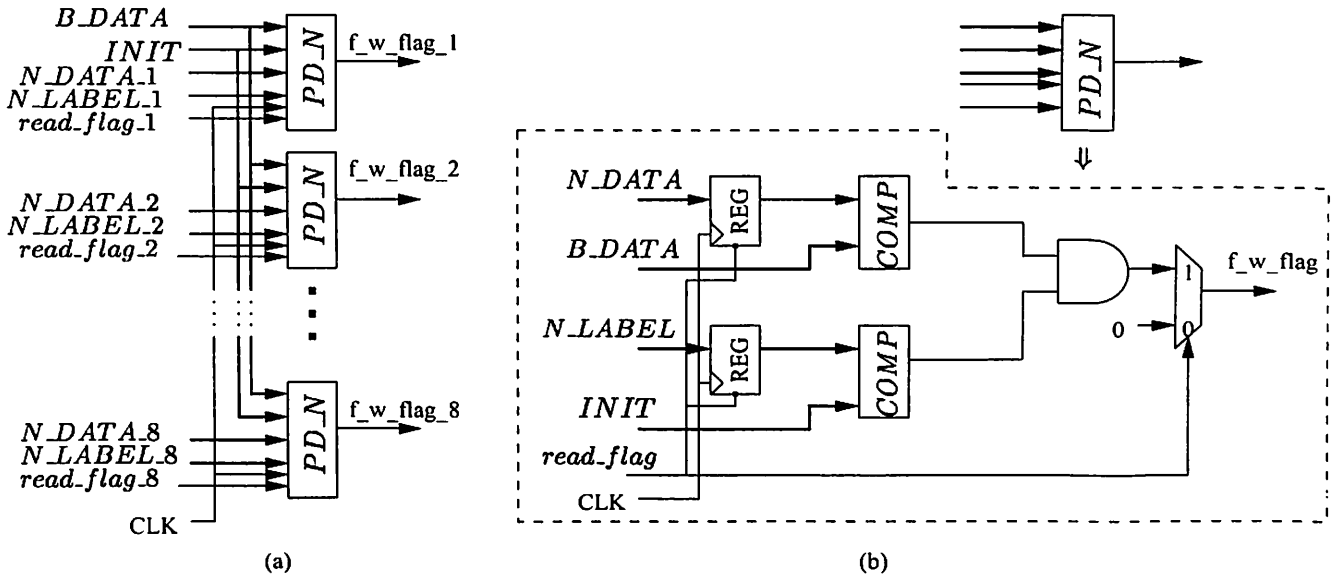


Figure 3.18: (a). The block diagram of the Plateau Detection Block. (b). Schematic diagram for PD_N block

Plateau Analysis Block (PA)

The schematic diagram shown in Figure 3.19 is used to detect whether the center pixel belongs to the non-minima plateau. The control signal plateau_analysis_flag is generated when the neighbor pixel data (N_DATA) is less than the center pixel data (B_DATA). It indicates that the center pixel belongs to the

non-minima plateau.

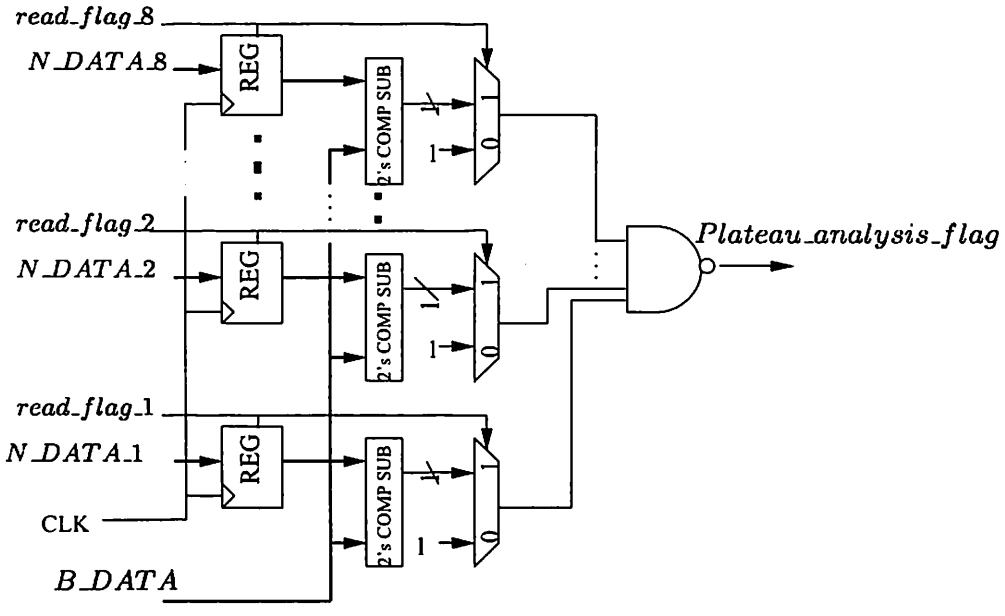


Figure 3.19: The Schematic diagram of the Plateau Analysis Block.

3.4.2 Simulated Flooding Process Circuit

The circuit for computing the simulated flooding process is shown in Figure 3.20. This circuit consists of two main functional blocks namely, (1) NS_AG meant for neighborhood and supporting pixels and (2) CNC meant for subsequent conditional neighborhood comparison. The FIFO is used in the recursive label propagation procedure. The NS_AG block generates the neighborhood pixel addresses (N_ADDR1 – N_ADDR8) and the supporting pixel addresses (S_ADDR1 – S_ADDR16) of the center pixel (B_ADDR) and loads the registers with the corresponding pixel and label values. The CNC block decides the labels of the neighboring pixels based on conditional comparisons.

The Finite State Machine (controller) FP_FSM generates the required control signals to operate this circuit. The control sequence of the above circuit is as follows.

Step 1: Reset L_M_addr_count. Initialize all flags to 0 and reset all functional blocks to 1.

Step 2: Read label in L_M from LOCAL_MINIMA_ RAM addressed by L_M_addr_count.

Set F_RST to 1.

Enqueue the L_M in FIFO.

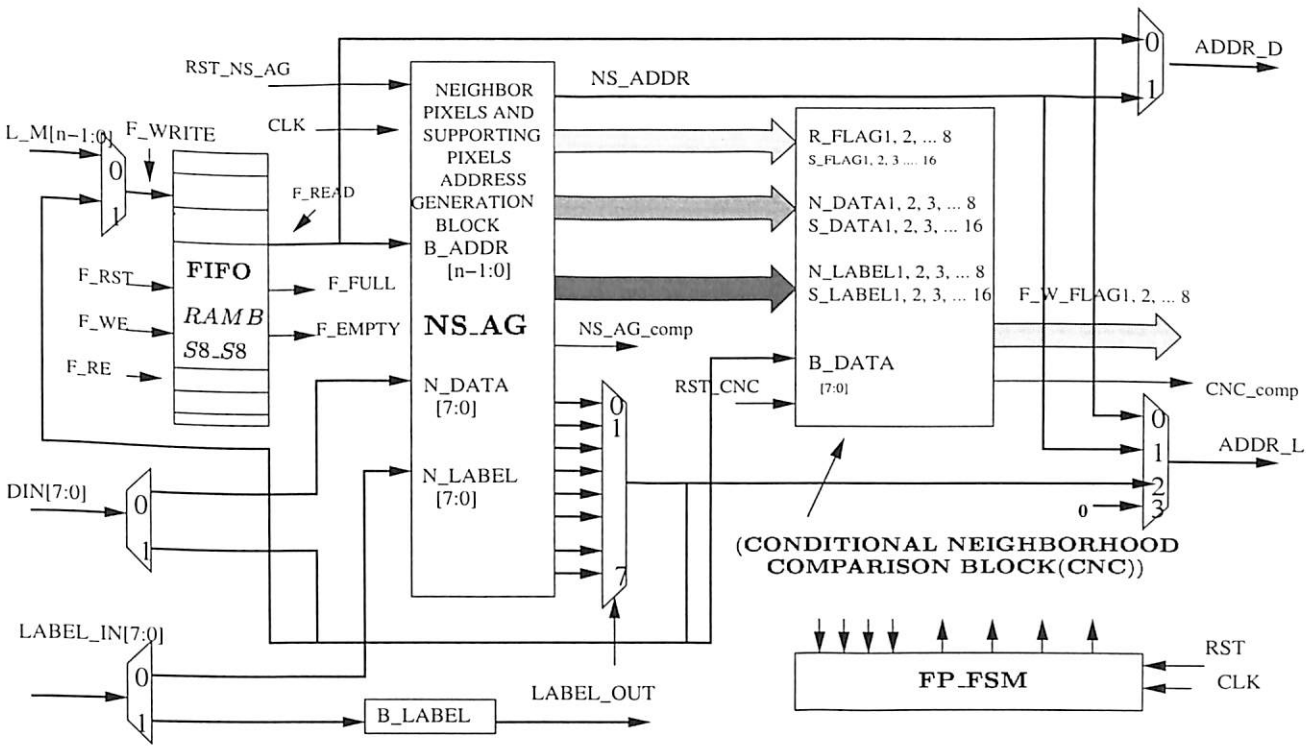


Figure 3.20: Circuitry for simulated flooding process

Step 3: Dequeue the value from FIFO.

Set RST_NS_AG to 0.

If NS_AG_comp is 1 **then** set RST_CNC to 0.

Step 4: **If** CNC_comp is 1 **then**

write center pixel label (B_LABEL) in LABEL_IMAGE_RAM pointed by corresponding neighborhood addresses (N_ADDR1–N_ADDR8).

Enqueue corresponding neighborhood addresses (N_ADDR1–N_ADDR8) in FIFO.

Step 5: **If** F_EMPTY is 1 then set F_RST to 1

else go to Step 2.

Step 6: Increment the L_M_addr_count.

If L_M_addr_count value is **not** equal to the maximum local minima **then** goto Step 2.

Step 7: Exit /* End of the flooding process */

The CNC module of the architecture is described as follows.

Conditional Neighborhood Comparison Block (CNC)

Figure 3.21 depicts the circuitry for conditional neighborhood comparison, which accommodates four 8-connected processing elements (labelled as PE8) and four 4-connected processing elements (labelled as PE4) involved with comparing the intensities of neighboring pixels (N_DATA1−N_DATA8) as well as supporting pixels (S_DATA1−S_DATA16) with that of the corresponding center pixel (B_DATA). The architecture of CNC block allows parallel operation of the four PE4 blocks as well as the four PE8 blocks. It may be mentioned here that Figure 3.3 effectively depicts the label propagation mechanism during the simulated flooding phase of the proposed algorithm. Now comparing Figure 3.3 and Figure 3.21, it is apparent that the four PE4 hardware blocks (vide Fig. 3.21) can simultaneously perform the conditional neighborhood comparison operation required for the 4-connected neighbors N_2, N_4, N_5, and N_7, of the center pixel C whereas the four PE8 blocks (of Fig. 3.21) can simultaneously carry out the conditional neighbor comparison at the four corner pixels N_1, N_3, N_6, and N_8 of Figure 3.3.

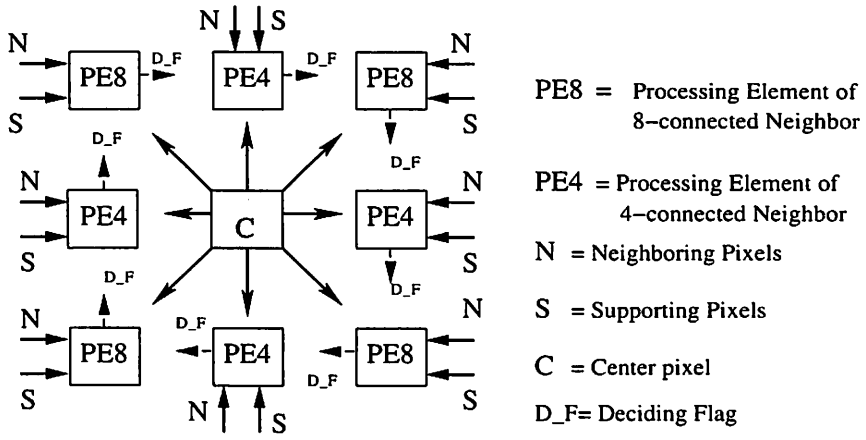


Figure 3.21: Circuitry for conditional neighborhood comparison

The hardware circuit for PE4 and PE8 are shown in Figures 3.22 and 3.23 respectively. The data bus loads the input data to a BUFFER inside a PE4 and PE8. The labeled C_N module block is used to detect the validity of the deciding neighboring pixel (N_DATA). The validity signal C_N_Flag is generated when the neighboring pixel (N_DATA) has its intensity value greater than that of the center pixel (C_DATA) and the neighboring pixel label (N_LABEL) is equal to NARM (not a regional minima). The blocks Four_con_comp and Eight_con_comp are used to decide the label of the neighboring pixel (N_DATA) based on conditional comparisons. A signal $load\_flag$ is generated when all the conditions is satisfied. This enables the state register to load the neighbor pixel label (N_LABEL) with the center

pixel label (C_LABEL). The performance of the conditional neighborhood comparison block is limited to 61.84MHz in XCV400HQ240

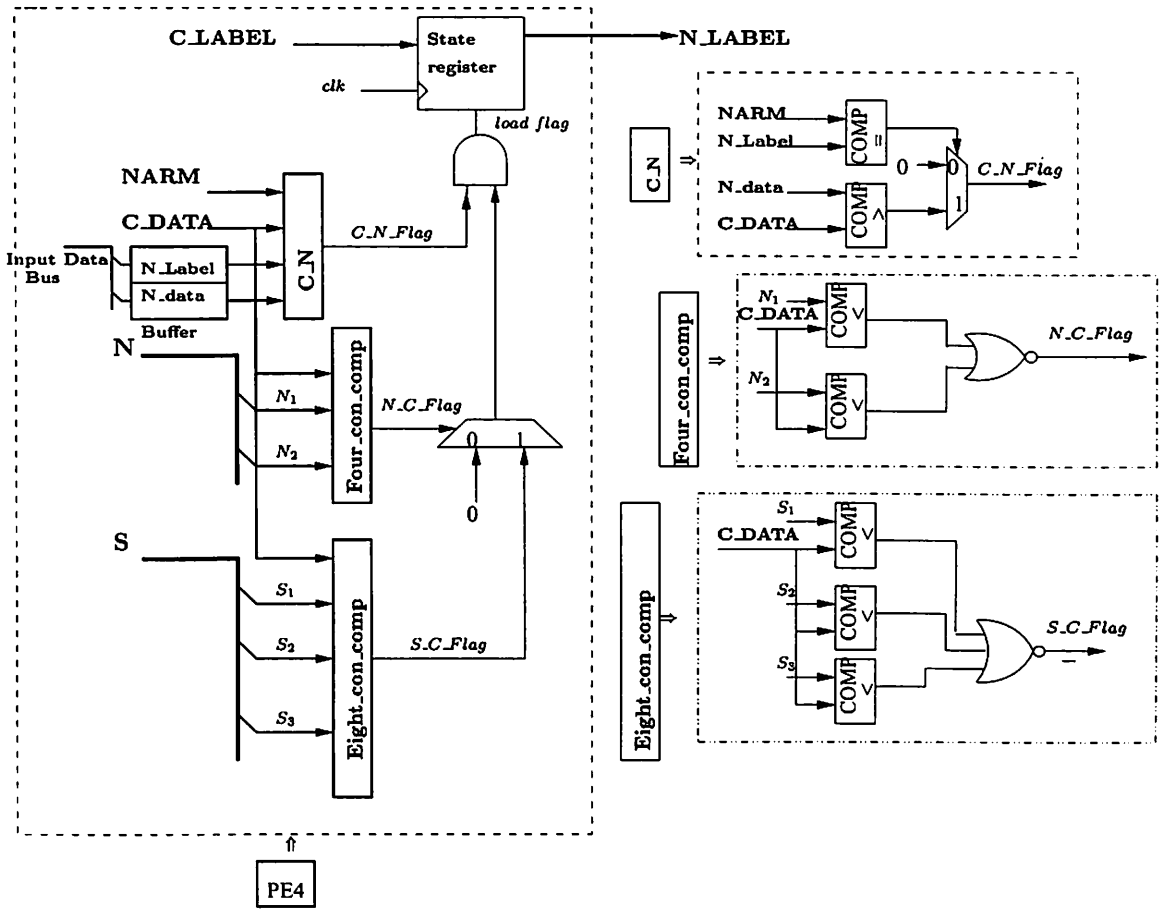


Figure 3.22: Circuitry for deciding the label of 4-connected neighbor pixel (PE4)

3.4.3 FPGA Implementation

FPGA [Tri94] provides an efficient platform for implementing a digital system with a short design turnaround time. The proposed watershed hardware architecture described in section 3.4.1 and 3.4.2 has been simulated by using Modelsim and synthesized under the **XILINX** ISE4.1i system [xil] targeted to Virtex FPGA series device (vide Appendix B). The proposed architecture has been simulated for various test images of size (30×30). An image of size greater than 30 × 30 requires more than one chip for its implementation. Due to limitations of the memory on the Virtex device, the FPGA implementation has been limited to an image of size 30 × 30. In this design the block RAMs are configured with 8-bit data lines and 9-bit address lines. Tests were run assuming a clock frequency of 50 MHz. The

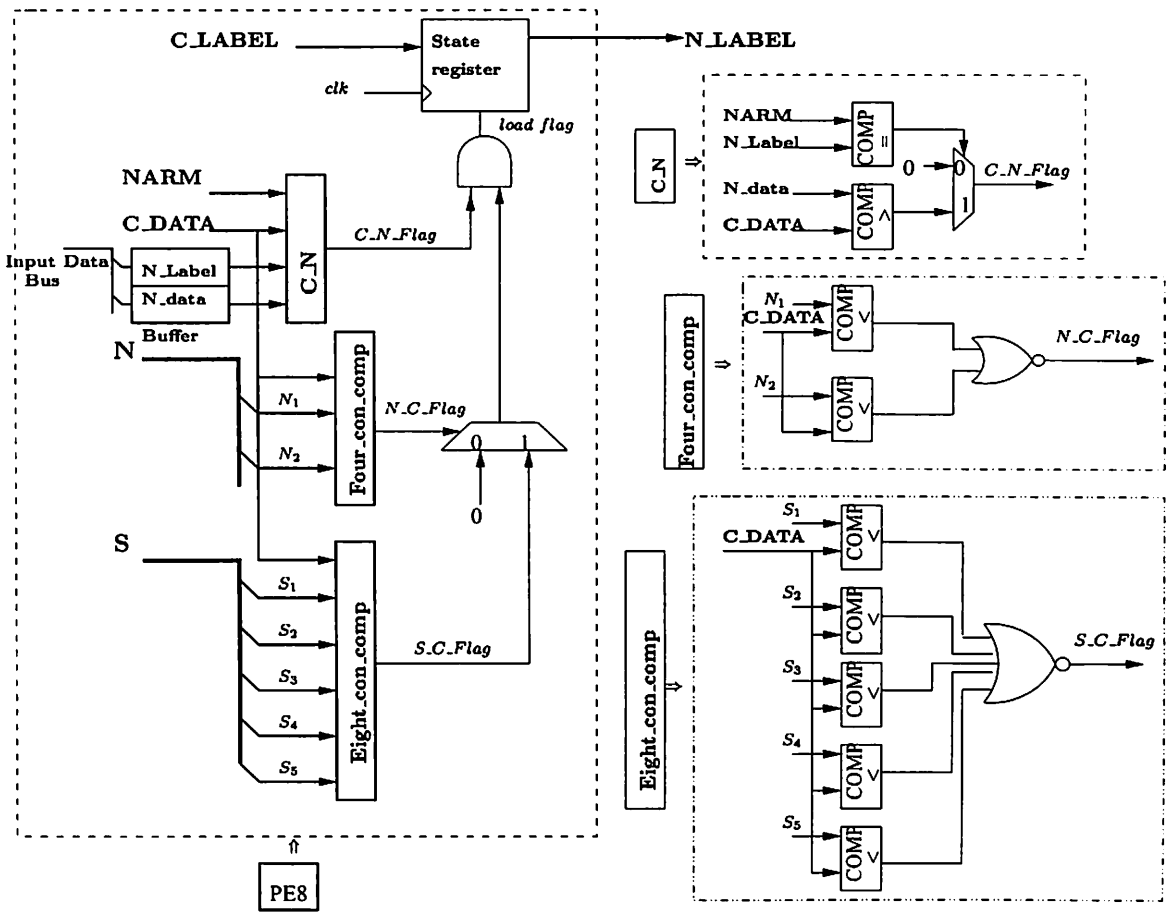


Figure 3.23: Circuitry for deciding the label of 8-connected neighbor pixel (PE8)

simulation results obtained for various test images are given in Table 3.4. From the simulation results, the hardware implementation is seen to be faster than the software version by an order of 3. The software results shown in Table 3.4 are achieved using Linux 9.0 environment on a Pentium IV 1.5 GHz processor system.

At this point, it should be mentioned that the proposed hardware architecture is not scalable. For, an increase in image size from M×M to N×N say, would merely entail a proportional increase in size of memory, namely three RAM blocks and two FIFOs (vide Figures 3.13, 3.14 and 3.20) in the hardware implementation of the proposed algorithm. However, representation of gray value of a pixel remains unchanged at 8 bits. Moreover, the individual steps of the proposed algorithm are processed in strict sequence regardless of the size of the image. Therefore, apart from the memory and associated address registers, no other change in the proposed hardware architecture is necessary. Total computation time would increase in accordance with an increase in image size.

Test Image (30 × 30)	Software Time (*millisecond*)	Hardware Time(*μsecond*)
Claire	16	48.16
Tennis	16	48.64
Heart	12	46.82
Akiyo	10	44.48
MRI Brain	20	53.2
Cermet	20	52.52

Table 3.4: Comparison between simulation time of hardware implementation and software implementation

The entire design has been simulated, synthesized at gate level and then implemented on the device XCV400-6HQ240. The HDL synthesis results of the proposed architecture is presented in Tables 3.6 and 3.5. The performance of the FPGA implementation is limited to 54.888 MHz in XCV400-6HQ240. The FPGA design summary of the entire architecture is accommodated in Table 3.7. The design utilizes almost 64% of CLBs and 25% of Registers.

Cell Usage			
#BELS	6101	# RAMs	5
BUF	4	RAMB4_S16	1
GND	1	RAMB4_S8	2
LUT1	580	RAMB4_S8_s8	2
LUT2	1118	# Tri-States	9
LUT3	2676	BUFT	9
LUT3_D	1	# Clock Buffers	1
LUT4	1151	BUFGP	1
LUT4_D	4	# IO Buffers	137
MUXCY	70	IBUF	136
muxf5	371	OBUF	1
MUXF6	54		
VCC	1		
XORCY	70		
# Flip Flops/Latches	2461		
FDC	296		
FDCE	54		
FDP	11		
FDRE	25		
LD	2105		

Table 3.5: HDL synthesis Report: Cell usage

Macro Statistics			
# FSMs	48	# Tristates	1
# Registers	18	9-bit tristate buffer	1
1-bit register	9	# Adders/Subtractors	9
8-bit register	2	9-bit adder	5
9-bit register	7	9-bit addsub	2
# Multiplexers	11	8-bit adder	2
9-bit 24-to-1 multiplexer	1	# Xors	400
8-bit 4-to-1 multiplexer	1	1-bit xor3	400
9-bit 8-to-1 multiplexer	3		
9-bit 4-to-1 multiplexer	6		

Table 3.6: HDL synthesis Report: Macro statistics.

Hardware Elements	**No. of Elements used**	**Percentage of Utilization**
Number of Slices	3,110 out of 4,800	64%
Total Number Slice Registers	2,481 out of 9,600	25%
Number used as Flip Flops	386	
Number used as Latches	2,095	
Total Number 4 input LUTs	5,012 out of 9,600	52%
Number used as LUTs	4,889	
Number used as a route-through	123	
Number of bonded IOBs	137 out of 166	82%
Number of TBUFs	9 out of 4,960	3%
Number of Block RAMs	5 out of 20	25%
Number of GCLKs	1 out of 4	25%
Number of GCLKIOBs	1 out of 4	25%
Total equivalent Gate count for Design:		126,544
Additional JTAG Gate count for IOBs:		6,624
Timing Summary:		
Minimum Period:	18.219ns	
(Maximum Frequency:)	54.888MHz	
Maximum combinational Path Delay:	16.489ns	
Maximum Net Delay:	7.977ns	

Table 3.7: Design statistics for the entire architecture.

3.5 Conclusions

In this chapter, a new improved technique of simulated flooding based watershed algorithm has been described. The proposed algorithm exhibits equivalent performance at reduced hardware complexity while compared to a conventional watershed algorithm [BM93]. The reduced hardware complexity results from use of a single queue instead of 256 queues required by the conventional algorithm to decide the labels of the neighbors of a given center pixel based on conditional neighborhood comparisons. Computational complexity of the proposed algorithm has been analyzed in detail. Moreover, a quantitative measure of accuracy of the segmentation results produced on various images by the algorithm has been provided. Apart from demonstrating its satisfactory performance on a number of images, this chapter has also provided a prototype hardware architecture for an effective realization of the proposed algorithm. The architecture has been synthesized in an appropriate FPGA environment. Also, the relevant design statistics of the FPGA implementation of the architecture are provided.

A drawback of the flooding-based watershed algorithm proposed in this chapter is that no synchronization in label propagation is possible in a non-minimum plateau which has more than one closely located regional minima. It leads to consequent degradation in performance. To prevent it, a hillclimbing method due to Moga [Mog97] can be employed. However, it is computationally costly. The next chapter describes a new hillclimbing technique which improves upon Moga's algorithm by doing away with the inherently time-consuming lower complete transformation involved in [Mog97].

Chapter 4

A Fast Hillclimbing-based Watershed Algorithm and its Prototype Architecture

This chapter presents a new watershed transform based on hillclimbing technique. Section 4.1 discusses about the conventional hillclimbing technique due to Moga [Mog97]. The next section gives a detailed discussion of the proposed hillclimbing technique. Section 4.3 analyzes the computational complexity of the principal steps of the proposed algorithm. Section 4.4 presents the simulation results of running the proposed algorithm and the conventional algorithm on different test images for comparison. In addition, this section evaluates the quality of segmentation produced by the proposed algorithm in a quantitative manner. An FPGA-based prototype architecture has been developed to implement the proposed algorithm involving moderate hardware complexity, and its implementation results are presented in section 4.5.

4.1 A Conventional Algorithm based on Hillclimbing Simulation

The original algorithm starts by detecting and labeling the initial seeds, that is, the minima of the gradient image, which characterize the regions of interest in a given image. The latter are defined as connected plateaus of pixels in the gradient image which do not have neighboring pixels of lower gray level (plateau of minima). Starting from the minima, region growing is then performed. Thus, the non-labeled pixels are assimilated into different components in an increasing order of gray levels. A characteristic of this flooding is that waves always progress upward, or are synchronized inside a plateau (as illustrated in Figure 4.1). Inside the flat areas, which are not yet labeled and are called plateaus of non-minima, components progress synchronously, such that they incorporate equal extents

within the plateau. Consequently, a pixel on such a plateau is labeled along the shortest path completely included in the plateau to a lower downward brim of the plateau.

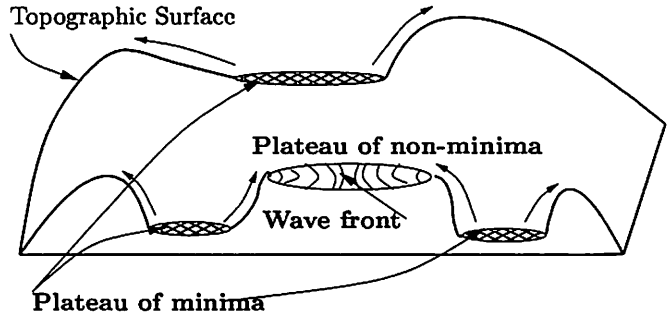


Figure 4.1: Flooding by hillclimbing.

A sequential watershed transform, which does not construct watershed lines, has the drawback of being scanning order dependent and hence, of producing inaccurate results in some cases. To overcome this problem, Meyer has introduced a lower-complete transform computation [MB90]. From two images, namely the gray scale image F and the lower distance image d (vide definition 3.7), a lower-complete image can be defined to include the ordering relations imposed by gray level and lower distance. This image is known as *lower-complete* [Mog97] and it is defined as follows.

Definition 4.1 (Lower-Complete Transform) *The mapping* $l(p) = \lambda * F(p) + d(p)$, $\forall p \in D_F$, *is called the lower-complete transformation of an image* F *based on the lower distance image* d.
Note that $\lambda > d(p)$, $\forall p \in D_F$.

Some representative values for λ are $\max_{p \in D_F} \{d(p)\}+1$, $\max_{0 \leq h \leq H} \{Histogram[F(h)]\} + 1$, where $Histogram$ is an array used to store the number of pixels with the gray level h for each gray level $h \in \{0, 1, \ldots, H\}$ in the image F. Another possible value of λ is $N \times M+1$, where $N \times M$ is the size of the image F. Note that in a lower-complete image l, every pixel $l(p)$ has a lower neighbor, except for the minima. This can be observed in Figure 4.2 which illustrates an example of a lower-complete transformation.

Lemma 4.1 *If* $F(p_1) < F(p_2)$ *then* $l(p_1) < l(p_2)$.

Proof— As $F(p) \in Z^+$, $F(p_1) < F(p_2)$ implies that $F(p_1) + 1 \leq F(p_2)$.

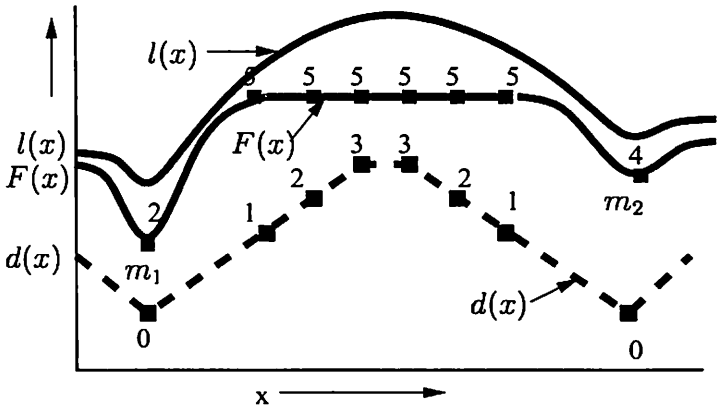


Figure 4.2: Lower complete transformation

According to the definition of λ, $d(p) < \lambda, \forall p \in D_F$, and $\lambda \geq 1$. Thus

$$\begin{aligned} l(p_1) &= \lambda * F(p_1) + d(p_1) < \lambda * F(p_1) + \lambda \quad \text{or,} \\ &< \lambda * (F(p_1) + 1) \end{aligned} \tag{4.1.1}$$

Now, as $F(p_1) < F(p_2)$,

$$\lambda * (F(p_1) + 1) \leq \lambda * F(p_2) \leq \lambda * F(p_2) + d(p_2) = l(p_2) \tag{4.1.2}$$

Therefore, from 4.1.1 and 4.1.2, $l(p_1) < l(p_2)$. □

Lemma 4.2 *If* $F(p_1) = F(p_2)$ *and* $d(p_1) < d(p_2)$ *then* $l(p_1) < l(p_2)$.

Proof— As $\lambda \geq 1$, the fact $F(p) \geq 0$ implies $0 \leq \lambda * F(p_1) = \lambda * F(p_2)$.

Moreover, $0 \leq d(p_1) < d(p_2)$ implies $\lambda * F(p_1) + d(p_1) < \lambda * F(p_2) + d(p_2)$. Hence, $l(p_1) < l(p_2)$.

□

Lemma 4.3 *If* $l(p_1) = l(p_2)$ *then* $F(p_1) = F(p_2)$ *and* $d(p_1) = d(p_2)$

Proof— The fact $l(p_1) = l(p_2)$ implies $\lambda * F(p_1) + d(p_1) = \lambda * F(p_2) + d(p_2)$, which in turn implies $d(p_1) - d(p_2) = \lambda * (F(p_2) - F(p_1))$. However, as $0 \leq d(p_1) < \lambda$, and $-\lambda < -d(p_2) \leq 0$ it signifies that $-\lambda < d(p_1) - d(p_2) < \lambda$ leading to $-\lambda < \lambda * (F(p_2) - F(p_1)) < \lambda$. Next, $\lambda \geq 1$ implies $-1 < F(p_2) - F(p_1) < 1$. Therefore, from $\Rightarrow F(p_2) - F(p_1) = 0$ or, $F(p_2) = F(p_1)$ one can conclude $d(p_1) = d(p_2)$. □

It is evident that on a lower-complete image, any pixel p can possibly have one or more neighbors with lower values, of which the one with the minimum value may be referred to as the *steepest lower-complete* (slc) neighbor of p, and denoted as slc(p). It is not however, necessarily unique. Flooding process is applied to a lower-complete image [MB90], in which the shortest paths actually correspond to the lines of the steepest slope. Moreover, every non-minima pixel is reached by flooding from its steepest lower-complete neighbor. Consequently, flooding should progress only between two neighboring pixels whose lower-complete transformed values satisfy the ordering relation involved in propagation of labels. Once a pixel has received a label, it is incorporated in the region it belongs to. In this algorithm, regions grow independently around their seeds, namely, the regional minima. More specifically, labels of minima are propagated from a candidate pixel upward to the higher neighboring pixels which do not have any neighbor with value lower than that of the given candidate for region growing. This bottom-up method is known as *hillclimbing simulation*. Based on the above definitions, the two major phases of the conventional hillclimbing simulation [Mog97] are (a) minima detection and labeling, and (b) simulated flooding, which are explained in the following sections.

4.1.1 Minima Detection and Labeling

Flooding is performed in the lower-complete image, after the regional minima have been detected and labeled. Consequently at this stage, a composite procedure is adopted to label the connected plateaus. For all other non-minima pixels, which have neighbors of lower gray level than themselves, the lower distance is evidently 1. It can be observed in Figure 4.3 that pixels within plateaus of minima (vide definition 3.5) have only neighboring pixels of higher or equal altitude. However, some pixels belonging to a plateau of non-minima (vide definition 3.6) have, additionally, neighboring pixels of lower altitude.

From the definition of an inner pixel (vide definition 3.4), it is clear that one pixel wide plateaus having lower neighbors would not be classified as plateaus of non-minima pixels. In such cases, motion obviously follows the steepest slope, and no lower distance needs to be computed. Therefore, a one pixel wide connected set of non-minimum, as exemplified by S_1 in Figure 4.3, is not a plateau of non-minima. Both the non-minimum pixels have lower neighboring pixels.

In the original algorithm [Mog97], a FIFO queue has been employed throughout the local minima detection and flooding processes. In this algorithm, flooding is performed on a lower-complete image, which can be derived by carrying out the operations, namely Neighborhood Inquiry, Plateau

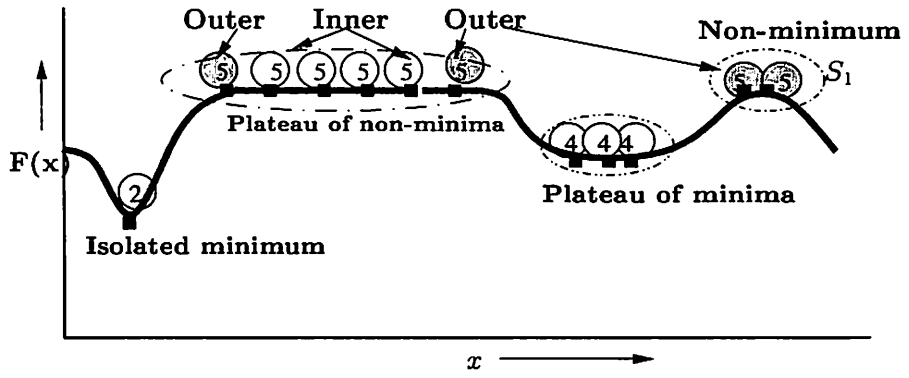


Figure 4.3: Plateau of minima and non-minima.

Analysis and Compute Lower Distance. First, a special label NARM (Not A Regional Minima) is initially assigned to every pixel in the labeled output image. In addition, the MAX_DIST (= ∞) value is set throughout the lower distance image, and a flag image is also initialized with flag value NOT_VISITED. The flag image contains the information about whether pixels of the label image are VISITED or NOT_VISITED. Since both the computation of the lower distance for minima, and the process of detection and labeling of regional minima deals with the connected plateaus of pixels, the latter ones are explored, when an inner pixel is encountered during the raster scan of the input gradient image. The inner pixels act as the protruding points into a connected plateau. While visiting a plateau, if an outer pixel is detected, the plateau, labeled so far as one of regional minima, becomes transformed into a plateau of non-minima, and the outer pixel is stored. Otherwise, it is correctly labeled as a plateau of minima. After the whole plateau has been visited, if some outer pixels have been encountered, another breadth-first scan is carried out, starting from all the collected outer pixels, to reset the label of the plateau to NARM, and compute the lower distances. After completion of the raster scan procedure, a lower complete image is generated according to definition 4.1, where λ becomes the maximum pixel value within the lower distance image. The image after labeling the regional minima and the distance image are depicted in Figure 4.4. The pseudocode of the local minima detection and labeling of minima is given below.

1: **Procedure 4.1** *Local_Minima_Detection(F, L, d)*
2: **Input**: Gradient image F. $\{F : D_F \rightarrow \mathbb{N}\}$
3: **Output**: Label image L, distance image d and flag image $flags$.
4: $current\_label \leftarrow 0$; {/* initialize the Current label */ }

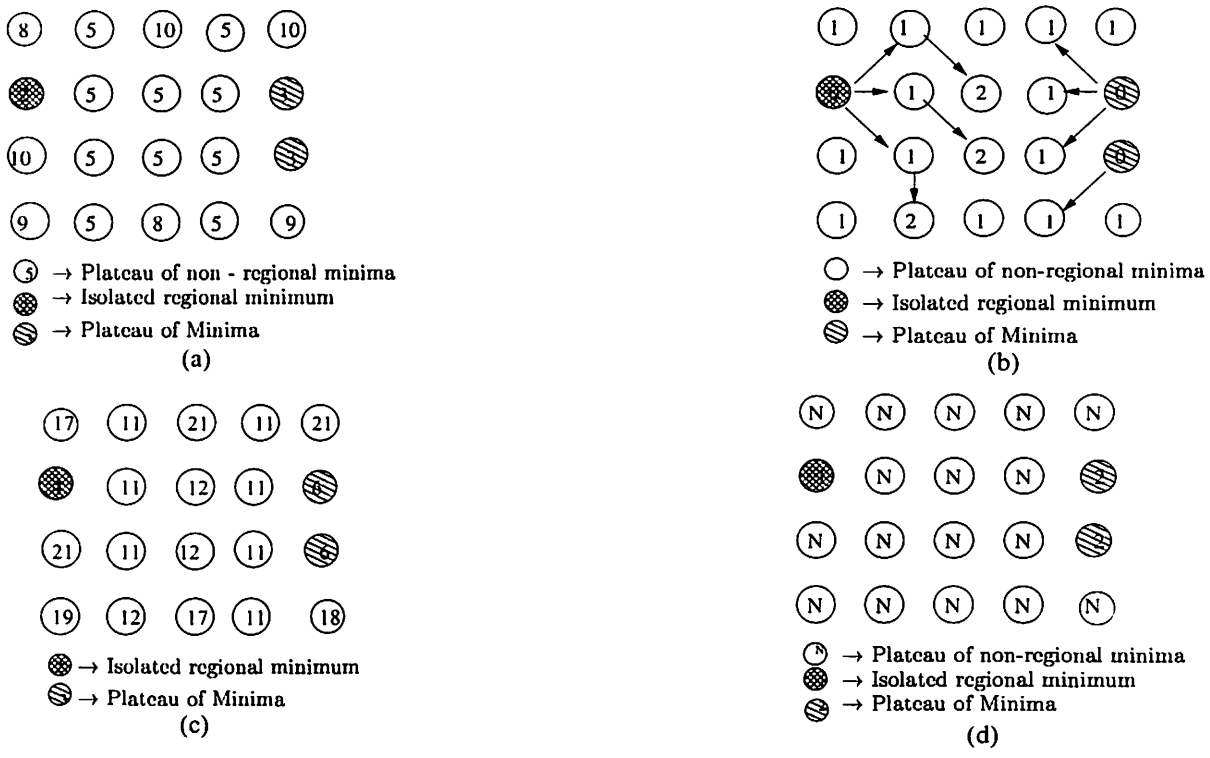


Figure 4.4: (a) The sample image; (b) Lower distance image; (c) Lower-complete image (λ = 2); (d) Output label image after regional minima detection and labeling (N = Not A Regional Minima).

5: $\lambda \leftarrow 1$;
6: **for all** ($p \in D_f$ with $flags(p)$ = NOT_VISITED)(Raster scan) **do**
7: Neighborhood_Inquiry(p, F, $pixel\_type$); /* Procedure for inquiring the neighborhood pixels of p */
8: **if** $pixel\_type$ = MINIMUM {/* it is an *isolated minimum* */} **then**
9: $L(p) \leftarrow current\_label$ ++;
10: **else if** $pixel\_type$ = ON_PLATEAU {/* a plateau is detected */} **then**
11: Plateau_Analysis(p, F, L, d, $flags$, $current\_label$, λ); /* procedure for detection and labeling of plateau */
12: $flags(p) \leftarrow$ VISITED;
13: **end if**
14: **end for**
15: /* End of procedure Local_Minima_Detection */

Procedure Neighborhood_Inquiry used in the procedure Local_minima_detection as same as procedure

3.3.1. Procedure Plateau_Analysis used in the procedure Local_Minima_Detection may be given with the help of pseudo-code as follows.

1: **Procedure 4.1.1** *Plateau_Analysis*(p, F, L, d, $current\_label$, $flags$, λ) /* *procedure for plateau detection and analysis* */
2: Input: p, F, L, d, $current\_label$ and $flags$, λ.
3: Output: L, d, $current\_label$ and λ.
4: $L(p) \leftarrow current\_label$;
5: FIFO_INIT(Q); $nouters \leftarrow 0$; /* initialize the FIFO queue and nouters, the number of outer pixels */
6: FIFO_ADD(p, Q); /* enqueue the pixel p */
7: **while** fifo queue Q not empty **do**
8: $p \leftarrow$ FIFO_REMOVE(Q);
9: **for all** $q \in N_G(p)$ **do**
10: **if** (($L(q)$ = NARM) and ($F(p) = F(q)$)) {/* plateau detection */} **then**
11: $L(q) \leftarrow current\_label$;
12: FIFO_ADD(q, Q);
13: $flags(q) \leftarrow$ VISITED;
14: **else if** ($F(q) < F(p)$) and ($d(p)$ = MAX_DIST) {/* pixel p is detected as an outer pixel */} **then**
15: $d(p) \leftarrow 1$; $Outers(nouters++) \leftarrow p$ /* add pixel p into the array Outers and assign a distance value 1 to it */
16: **end if**
17: **end for**
18: **end while**
19: **if** ($nouters$ = 0) {/* a minima plateau is detected */} **then**
20: $current\_label$ ++;
21: **else**
22: /* Non-minima plateau detected */
23: Compute_Lower_Distance($Outers$, $nouters$, $F(p)$, L, d, $flags$, λ); /* procedure for computing lower distances of the inner pixels in the non-minima plateau */
24: **end if**

25: /* End of procedure Plateau_Analysis */

Procedure Compute_Lower_Distance used in the procedure Plateau_Analysis may be described with the help of pseudo-code as follows.

1: **Procedure 4.1.1.1** *Compute_Lower_Distance(Outers, nouters, $F(p)$, L, d, $flags$, λ); /* procedure for lower distance computation of inner pixels */*
2: Input: $Outers$, $nouters$, $F(p)$, F, L, d, $flags$ and λ.
3: Output: d, L and λ.
4: STORE_FIFO_QUEUE($Outers$, $nouters$); /* Enqueue the outer pixels */
5: $time\_stamp \leftarrow 1$; $wave\_length \leftarrow$ queue_content;
6: **while** ($wave\_length > 0$) **do**
7: $time\_stamp$++;
8: **for** (k=0; k < $wave\_length$; k++) **do**
9: $p \leftarrow$ FIFO_REMOVE(Q);
10: **if** ($L(p) \neq$ NARM) **then**
11: $L(p) \leftarrow$ NARM
12: **end if**
13: **for all** ($q \in N_G(p)$ and $F(q) = F(p)$ and $d(q)$ = MAX_DIST) {/* for all inner pixels */} **do**
14: $L(p) \leftarrow$ NARM; $d(q) \leftarrow time\_stamp$
15: FIFO_ADD(q, Q);
16: **end for**
17: **end for**
18: $wave\_length \leftarrow$ queue_content;
19: **end while**
20: $\lambda \leftarrow \max(\lambda, time\_stamp)$; /* find the maximum distance */
21: /* End of the procedure Compute_Lower_Distance*/

4.1.2 Simulated Flooding

Another raster scan is used to perform the flooding process, in which a FIFO queue is initialized with the pixels within plateaus of minima detected in the previous procedure of local minima detection and labeling. These pixels have non-minima pixels in their neighborhood. An additional image *slc* is

used to store the altitudes of the steepest lower complete neighbors of the corresponding pixels. After completion of the raster scan, the flooding process is deemed to be over.

A candidate pixel p removed from the queue propagates its label to all its neighboring pixels q if there is an arc from p to q in the forest, that is, $slc(q) = l(p)$. Any new labeled pixel becomes a candidate, and it is inserted in the FIFO queue. When the queue of candidates is emptied, each pixel in the output image gets labeled and appended to a single connected component and the hillclimbing procedure stops. The set of non-overlapping components contributes a complete partition of the image. The output labeled image is illustrated in Figure 4.5. The pseudo-code of the simulated flooding process

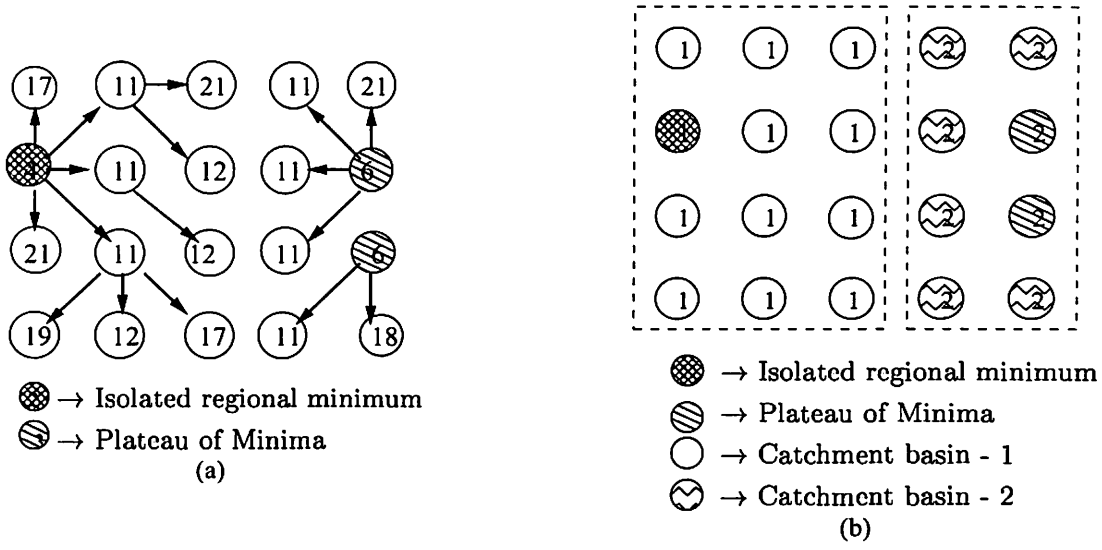


Figure 4.5: (a) Flooding the lower-complete image; (b) Output image of labels.

is as follows.

1: **Procedure 4.2** *Hillclimbing_Technique(L, l, slc, flags)*
2: **Input:** Label image L, lower-complete image l, steepest lower image slc and flag image $flags$.
3: **Output:** Label image L.
4: **for all** ($p \in D_F$)(Raster scan) **do**
5: **if** (L(p) < NARM and $\exists q \in N_G(p)$, $L(q)$ = NARM) {/* for each non-labeled neighboring pixel */} **then**
6: FIFO_ADD(p, Q);
7: **else**
8: $slc(p) \leftarrow \min\{\, l(q) | q \in N_G(p),\ l(q) < l(q)\}$ /* lowest neighbor of pixel p in the lower-

```
        complete image */
 9:   end if
10: end for
11: while (queue is not empty) do
12:   p ← FIFO_REMOVE(Q);
13:   for all (q ∈ N_G(p) and flags(q) ≠ NOT_VISITED) do
14:     if (L(q) = NARM and l(p) = slc(q)) then
15:       L(q) ← L(p);
16:       FIFO_ADD(q, Q);
17:     end if
18:   end for
19: end while
20: /* End of the procedure Hillclimbing_Technique */
```

Before going into the details of the proposed method, a relevant definition is given first.

Definition 4.2 (Steepest lower neighbor image) *The steepest lower neighbor image, denoted by sln, corresponding to a gray scale image F, is derived from the latter as follows. Each pixel $p \in D_F$ is replaced by its steepest lower neighbor, given by $sln(p) = min_{q \in N_G(p)} \{ F(q) \mid F(q) < F(p) \}$, if p is an outer pixel; otherwise, $sln(p) = F(p)$.*

4.2 Proposed Hillclimbing Simulation

In Moga's algorithm [Mog97], first the input gradient image has been transformed into a lower-complete image. This transformation involves O($m \times n$) multiplications and additions, where $m \times n$ is the size of the input image. The algorithm proposed in this chapter attempts to reduce the computational complexity in two ways. First, the lower complete image computation is avoided. Secondly, the plateau detection and its labeling is computed in one scan rather than two scans by the use of two queues. Flooding of the non-minima plateau pixels has been done by directly using the distance image. Also in Moga's algorithm, the raster scan operation has been employed twice, first in "Minima Detection and Labeling" process and next in "Simulated Flooding" process. During the second raster scan, minima and plateaus of minima, which have been labeled during the first raster scan, are stored in a FIFO

queue. In the proposed algorithm, however, starting from the minima, the recursive label propagation (flooding) is performed in breadth-first order using a single FIFO queue, a distance image and a steepest lower neighbor image *(sln)* which stores the steepest lower neighboring altitude. This steepest lower image is computed during the first raster scan (local minima detection and labeling process) rather than the following scan. Thus the neighboring pixels need not be accessed again. This improves the complexity of the algorithm. Moreover, during plateau analysis, the outer pixels are stored in an array *Outers*, which will be used in the procedure of computing the lower distance for subsequent label assignment. Amongst the pixels belonging to a plateau, it may be recalled that those having at least one neighbor with lower altitude are termed *outer* pixels while the others are called *inner* pixels. In Moga's algorithm, these outer pixels are first inserted in a FIFO queue, and then the lower distance computation has been carried out for inner plateau pixels. Time required for movement of pixels from *Outers* array to the FIFO queue can be diminished by employing another FIFO queue, where the outer pixels will be directly enqueued. The detailed description of the two major phases of the proposed hillclimbing technique is provided next.

4.2.1 Detection and labeling of Local Minima

In this procedure, a single raster scan has been employed, and two FIFO queues Q_{PD} (for plateau detection) and Q_{PA} (for plateau analysis) have been utilized as shown in Fig. 4.6. For each not yet labeled pixel p, its 4-connected or 8-connected neighborhood is inspected. Thus if all the neighbors are of higher gray level than p, then p is deemed to be an *isolated minimum*, and a label is assigned to it. Else, if this pixel belongs to a plateau, the plateau is scanned in a breadth-first order, and the visited pixels are labeled with the current label, and inserted into the queue Q_{PD} for detection of plateau. The examination always starts from an *inner* pixel which introduces the neighboring pixels of equal altitude in the list of candidates. A currently investigated candidate pixel may still qualify as an *inner* pixel; otherwise, it is an *outer* pixel. The lower distance of the outer pixel is 1, and inserted into queue Q_{PA} for analysis of plateau. After scanning the plateau, if the queue Q_{PA} is not empty (indicating that a non-minima plateau is detected), it is then necessary to compute the distance of *inner* pixels (from the outer pixels) and assign NARM (not a regional minimum) label to the plateau currently detected for subsequent flooding. Also the steepest lower neighboring altitude pixels are computed and stored in an additional image known as the steepest lower neighbor image, which has been defined above.

The regional minima detection and labeling in a sample image, its steepest lower neighbor image

and lower distance image are illustrated in Figures 4.7 and 4.8. The pseudo-code of the whole procedure of detection and labeling of local Minima is given below.

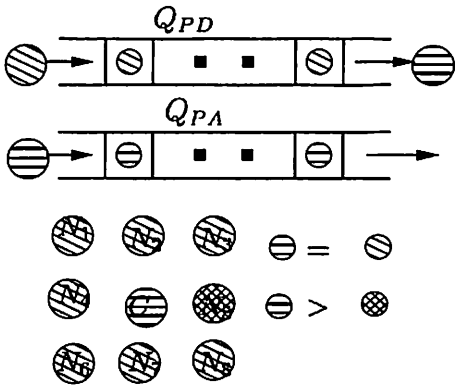


Figure 4.6: Local minima detection and Labeling Process

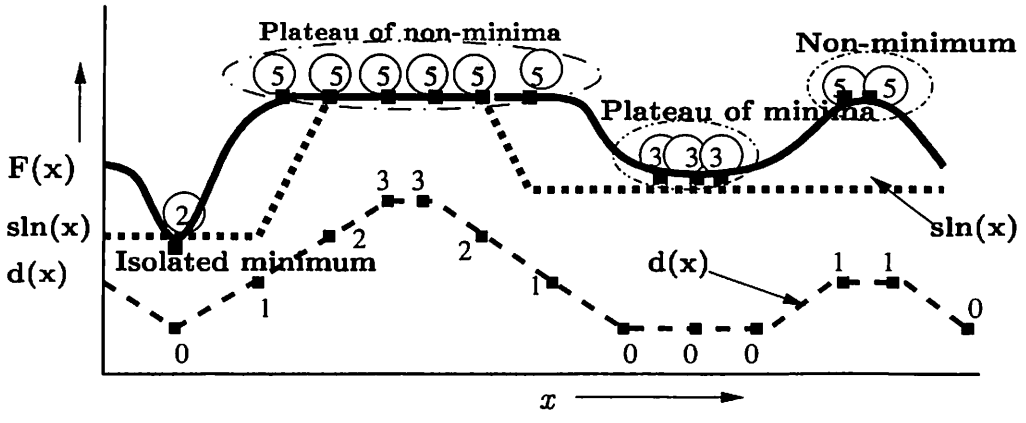


Figure 4.7: Lower distance and steepest lower neighbor of image F.

1: **Procedure 4.3** *LM_Det_Labeling(F, L, d, sl) /* procedure for detecting and labeling local minima */*

2: Input: Morphological multi-scale gradient image F. $\{F : D_F \rightarrow \mathbb{N}\}$

3: Output: Label image L, label image L, distance image d and steepest lower image sln. $\{L, d, sln$ $: D_F \rightarrow \mathbb{N}\}$

4: Let G be a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$ and $N_G(p)$ stands for the neighbors of a pixel p on the grid G.

5: INIT $\leftarrow -2$;{/* initialization value */}

6: NARM $\leftarrow -1$;{/* Not A Regional Minima */}

7: MAX_DIST $\leftarrow -1$;

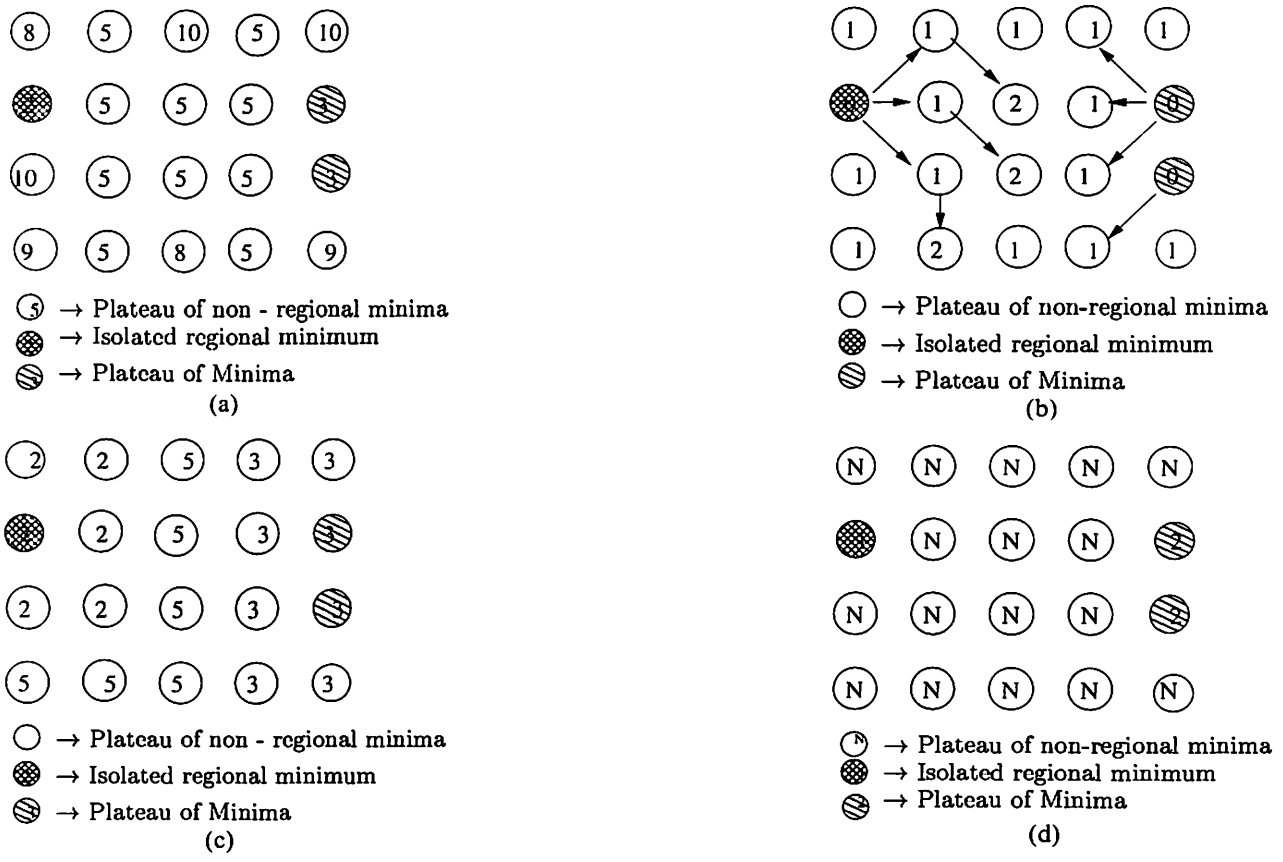


Figure 4.8: (a) The sample image; (b) Lower distance image; (c) Steepest lower neighbor image; (d) Output label image after regional minima detection and labeling (N = Not a Regional Minima).

8: $current\_label \leftarrow 0$ {/* initialize the Current label */ }

9: **for all** ($p \in D_F$) **do**

10: $L(p) \leftarrow$ INIT; $d(p) \leftarrow$ MAX_DIST; /* label image and distance image initialization */

11: **end for**

12: **for all** ($p \in D_F$ with $L(p)$ = INIT)(Raster scan) {/* for each pixel in the image which is not yet visited */} **do**

13: Neighborhood_Inquiry(p, F, $pixel\_type$); /* procedure for inquiring the neighborhood pixels of p */

14: **if** $pixel\_type$ = MINIMUM {/* It is a *isolated minimum* */} **then**

15: $L(p) \leftarrow current\_label$++;

16: **else if** $pixel\_type = ON\_PLATEAU$ {/* A plateau is detected */} **then**

17: Plateau_Analysis(p, F, L, d, $current\_label$); /* procedure for plateau detection and it:

labeling */

else if $pixel\_type$ = NON_MINIMUM {/* a non-minimum pixel is detected */} **then**

$L(p) \leftarrow NARM$;

end if

end for

/* End of procedure LM_Det_Labeling */

Procedure Neighborhood_Inquiry used in the procedure LM_Det_Labeling is the same as procedure 3.3.1. Procedure Plateau_Analysis used in the procedure LM_Det_Labeling given next however, differs from the corresponding procedure 3.3.2 of the proposed an improved flooding-based watershed algorithm described in the previous chapter as follows. Here, one needs to construct the steepest lower neighbor image and its lower distance image.

Procedure 4.3.1 *Plateau_Analysis(p, F, L, d, current_label) /* procedure for detection and labeling of plateau*/*

Input: p, F, L, d, $current\_label$.

Output: L, sl, $current\_label$.

$L(p) \leftarrow current\_label$;

FIFO_INIT(Q_{PD}); FIFO_INIT(Q_{PA});{/* initialize the FIFO Queues (Q_{PD}) and (Q_{PA})*/}

FIFO_ADD(p, Q_{PD}); {/* insert the pixel p into the queue Q_{PD} */}

while fifo queue Q_{PD} not empty {/* plateau Detection phase */} **do**

$p \leftarrow$ FIFO_REMOVE(Q_{PD}); {/* get candidate pixel p from queue Q_{PD} */}

min $\leftarrow F(p)$;

for all $q \in N_G(p)$ {/* inquire neighboring pixels of p */} **do**

if (($L(q)$ = INIT) and ($F(p) = F(q)$)) **then**

$L(q) \leftarrow current\_label$;

FIFO_ADD(q, Q_{PD});

else if (($F(q) < F(p)$) and ($d(p)$ = MAX_DIST)) {/* outer pixel detection */} **then**

$d(p) \leftarrow 1$;

FIFO_ADD(q, Q_{PA});

end if

if ($F(q) < F(p)$) **then**

min $\leftarrow F(q)$;

20: **end if**
21: **end for**
22: $sln(p) \leftarrow$ min; {/* collect steepest lower neighbor value into the sln image */}
23: **end while**
24: **if** FIFO QUEUE Q_{PA} is empty {/* a minima plateau is detected */} **then**
25: $current\_label := current\_label + 1$; /* increment the current_label value */
26: **else**
27: /* a non-minima plateau is detected */
28: Compute_Lower_Distance(L, d, Q_{PA}); /* procedure for computing the lower distances of the inner pixels */
29: **end if**
30: /* End of procedure Plateau_Analysis */

Procedure Compute_Lower_Distance used in the procedure Plateau_Analysis may be described with the help of pseudo-code as follows.

Compute Lower Distance

In this procedure, another breadth-first search is performed to reset the label of the plateau to NARM, and to compute the lower distance function. A wavefront, beginning at the outer pixels, drifts recurrently and exhaustively across the plateau. Pixels swept by a wave set the lower distance to the time-stamp of the wave, which is initially 1 and gets incremented at each iteration.

1: **Procedure 4.3.1.1** *Compute_Lower_Distance(L, d, $current\_label$, Q_{PA}); /* procedure for computing the lower distance */*
2: Input: L, d, $current\_label$ and Q_{PA}
3: Output: d and L.
4: $time\_stamp \leftarrow 1$;
5: $wave\_length \leftarrow$ FIFO_CONTENT(Q_{PA});
6: **while** ($wave\_length > 0$) **do**
7: $time\_stamp$++;
8: **for** (i=0; i < $wave\_length$; i++) **do**
9: $p \leftarrow$ FIFO_REMOVE(Q_{PA});
10: **if** ($L(p) \neq$ NARM) **then**

```
11:          L(p) ← NARM
12:        end if
13:        for all (q ∈ N_G(p) and L(q) = current_label and d(q) = MAX_DIST) do
14:          L(q) ← NARM; d(q) ← time_stamp
15:          FIFO_ADD(q, Q_PA);
16:        end for
17:      end for
18:    end while
19:    /* End of the procedure Compute_Lower_Distance */
```

4.2.2 Flooding of Minima

Another raster scan procedure is used to perform the flooding process, in which a FIFO queue is initialized with the pixels within plateaus of minima, as discussed in the previous section. Consequently, flooding should progress only between two neighboring pixels whose steepest lower complete neighbo· and distance values satisfy the flooding ordering relation. Once a pixel has received a label, it is correctly incorporated in the region it belongs to. The basis of the proposed flooding order is established by the following theorem.

Theorem 4.1 *$p \succ q$ (to be read p is flooded from q) in the grayscale image F whose lower distance image and steepest lower neighbor image are d and sln respectively if*

$$F(q) = F(r) \mid \{r \in N_G(p)\} \wedge \{sln(p) = F(r)\} \tag{4.2.1}$$

else, $p \succ q$ if

$$\{F(p) = F(q)\} \wedge \{sln(p) = F(q)\} \wedge \{d(p) = d(q) + 1\} \tag{4.2.2}$$

Proof of the above theorem is similar to that of theorem 3.1.

Proof— (proof of 4.2.1) Consider a pixel $q \in N_G(p)$ such that q satisfies the condition 4.2.1. Flooding always proceeds in increasing order of gray levels. Thus, considering flooding at level $F(q)$, it is either true that p is not yet flooded and it will be flooded by q; else, it has been already flooded from another neighbor $r \in N_G(p)$ $(r \neq q)$. If the second case holds, then the gray levels higher than the gray level $F(q)$ will be flooded only after the gray level $F(q)$ has been flooded. Thus $F(r) \leq F(q)$. However,

as q satisfies (4.2.1), it happens to be the neighbor of p with the lowest surrounding gray level. Thus, $sln(p) = F(q) = F(r)$. Hence, pixel p is flooded from q or r, depending on which one is scanned first. however, flooding always occurs from the surrounding neighbor with the lowest gray level, although this neighbors may not be unique.

(proof of 4.2.2) The lower distance (vide definition 3.7) is a measure of closeness of a pixel within a plateau to its lower brim. It also represents the time taken by the wave which advances from the closest lower brim of the plateau and floods the pixel. Thus, if pixel p on a plateau has its lower distance $d(p) > 1$, it will be flooded from a pixel which has been flooded by a wave at time $d(p) - 1$.

To prove (4.2.2) by contradiction, suppose p inherits its label from another neighbor r with lower distance $d(r) > d(p) - 1$. It means that the wave from r reaches the pixel p at time $d(r) + 1$ to find p without a label. From the hypothesis, $d(r) + 1 > (d(p) - 1) + 1$ or $d(r) + 1 > d(p)$. That is, two wavefronts which reach p along paths of shortest length fail to immerse the plateau at same rate. Hence, (4.2.2) is established by contradiction.

□

In this procedure, only one FIFO queue data structure Q_F has been used for recursive label propagation of the candidate pixels. First, one stores the seed pixels in the queue Q_F; next, labeling starts as follows. A candidate pixel p is removed from the queue Q_F, and its label is assigned to each of its neighboring NARM pixel q, which has a higher altitude. If the neighboring pixel q has the same altitude as p, then the distance image needs to be made use of to determine the label of q. Only those neighboring pixels, whose lower distance is one greater than that of the candidate pixel, get labeled. Any new labeled pixel becomes a candidate, and it is inserted in queue Q_F. This process is then repeated until the queue becomes empty. Flooding process in a sample image and its corresponding output image of labels are illustrated in Figure 4.9. The pseudo-code of the flooding process is given below.

1: **Procedure 4.4** *Flooding_Process(F, L, sln)*
2: Input: Multi-scale gradient image F, label image L and steepest lower neighbor image sln.
3: Output: Label image L.
4: FIFO_INIT(Q_F); {/* initiation of recursive label propagation */}
5: **for all** ($p \in D_F$ and $L(p) \neq$ NARM) **do**
6: FIFO_ADD(p, Q_F); /* initialize the FIFO queue with local minima */
7: **end for**

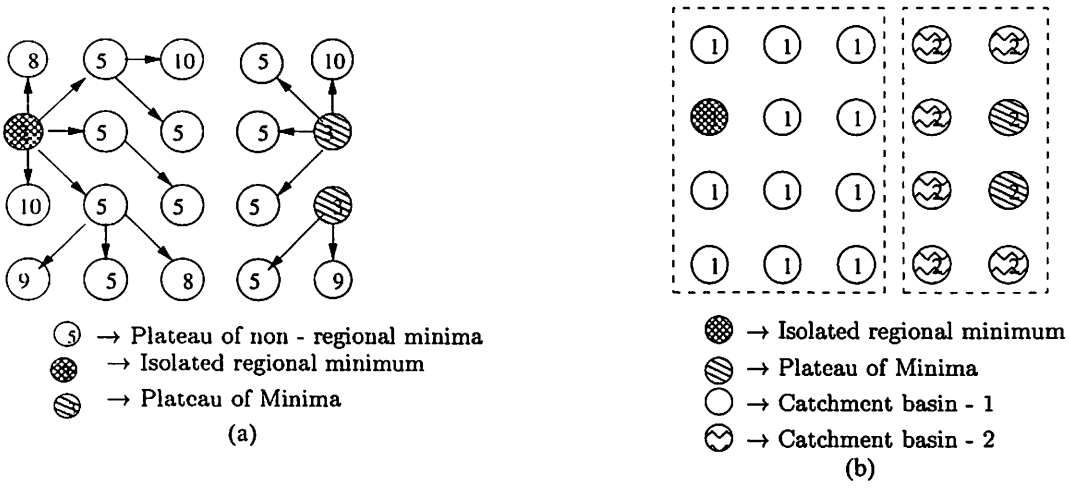


Figure 4.9: (a) Flooding the input sample image; (b) Output image of labels.

8: /* Start flooding */
9: **while** FIFO QUEUE Q_F is not empty **do**
10: $p \leftarrow$ FIFO_REMOVE(Q_F);
11: **for all** ($q \in N_G(p)$ and $L(q)$ = NARM) **do**
12: **if** ($sln(q) = F(p)$) and (d(p) = 1) and ($d(q) = d(p) + 1$) {/* label the non-labeled neighboring pixels $q \in N_G(p)$ based on the flooding order (vide theorem 4.1) */} **then**
13: $L(q) \leftarrow L(p)$;
14: FIFO_ADD(q, Q_F); /* enqueue the new labeled pixel q */
15: **end if**
16: **end for**
17: **end while**
18: /* end of procedure Flooding_Process */

4.3 Complexity Analysis

Let $n = M \times N$ be the dimension of the image F whose domain is denoted as $D_F \subset \mathbb{Z}^2$, and G is a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$. Let N_G stand for the neighborhood pixels on the grid G.

Minima detection and its labeling

Let p_1, p_2, p_3, ..., p_P be P plateaus, each p_i having n_i pixels and P_N be the detected non-minima plateaus.

1. The number of comparisons for plateau detection and plateau analysis is $\sum_{i=1}^{P} n_i * N_G$.
2. The number of comparisons for lower distance computation and non-minima plateau labeling is $\sum_{i=1}^{P_N} n_i * N_G$.

Total number of comparisons for minima detection and its labelling is then

$$LMD = (\sum_{i=1}^{P} n_i + \sum_{i=1}^{P_N} n_i) * N_G \approx n \tag{4.3.1}$$

Flooding process

Let l_1, l_2, l_3, ..., l_L be the L local minima plateaus in the image G; moreover, let l_j have m_j pixels which have at least one NARM neighborhood pixel. The total number of comparisons for the entire label propagation (in the worst case) is

$$LP = n + (\frac{1}{P_N} \sum_{i=1}^{P_N} n_i) * (\sum_{j=1}^{L} m_j * N_G) \simeq 2n \tag{4.3.2}$$

Total number of comparisons for the entire watershed computation (on an average case) is

$$LMD + LP \approx 3n \approx O(n) \tag{4.3.3}$$

Analysis of Conventional Hillclimbing Technique [Mog97]

The number of comparisons for minima detection and flooding process (on an average) is $3n$. In addition, the lower-complete transformation involves n multiplications and n additions. Thus the computational complexity of the conventional watershed algorithm (on an average) is

$$= 3n \text{ (comparisons)} + n \text{ (multiplications)} + n \text{ (additions)} \tag{4.3.4}$$

From the above analysis (4.3.3 and 4.3.4), the proposed algorithm requires only about $3n$ comparisons as against $3n$ comparisons, $O(n)$ multiplications and $O(n)$ additions for the conventional algorithm.

4.4 Experimental Results

As discussed above, a considerable reduction in the computational complexity has been obtained. To avoid over-segmentation, a morphological multi-scale gradient [Wan97] image with different threshold values are used as the input for computing the watershed transform. The improved algorithm has been implemented on a Pentium IV 1.5 GHz computer under LINUX 9.0 environment and applied on various test images to verify its effectiveness. The resultant watershed images are same for both the algorithms. Comparison of the computation times of both algorithms is given in Table 4.1. The improved algorithm is faster and more efficient than the existing algorithm. Also, the segmentation results are obtained for different values of the threshold h required as a parameter in construction of the multi-scale gradient [Wan97] image. Figure 4.10 shows that an increase in h leads to a reduction in the number of regions. Variation of simulation time with the values of h is plotted in Figure 4.11. Results of running both Moga's algorithm [Mog97] and the proposed algorithm on Table Tennis, Cermet, MRI Brain, Washington_ir, Steel Fissure and Blood cells images are shown in Figures 4.12, 4.13, 4.14, 4.15, 4.16 and 4.17 respectively, which indicate that the watershed images produced by both the algorithms are almost identical.

Test Image	Threshold value(h)	Number of homogeneous regions	Time (*Sec*)	
			Moga's [Mog97] Technique	Proposed Technique
Claire (352×288)	10	17	0.092	0.075
Tennis (352×288)	13	22	0.095	0.072
Heart (256×256)	8	14	0.083	0.065
Akiyo (164×144)	14	22	0.031	0.024
MRI Brain (256×256)	12	38	0.072	0.055
Cermet (256 × 256)	30	61	0.081	0.064
Washinton_IR (250×250)	16	46	0.098	0.073
Steel Fissure (290×369)	15	10	0.105	0.091
Blood Cells (212×353)	26	20	0.078	0.065

Table 4.1: Comparison between simulation time (in seconds) of Moga's Hillclimbing technique [Mog97] and the proposed Technique.

Following the region-based approach [HD95] stated in section 3.3.1 to quantitatively evaluate the performance of a segmentation algorithm, the performance measure P of the proposed algorithm evaluated for various test images is provided in Table 4.2, which shows that the homogeneous regions are

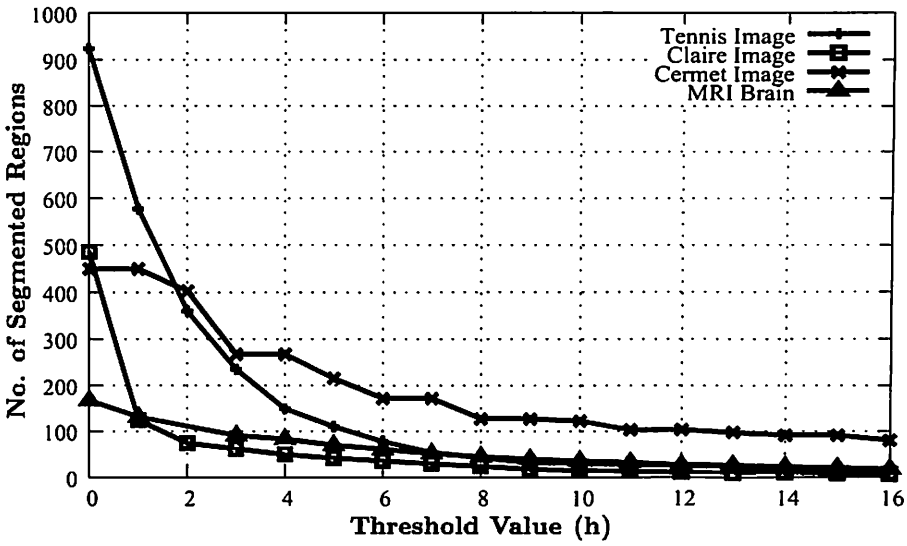


Figure 4.10: Number of regions for different threshold value (h).

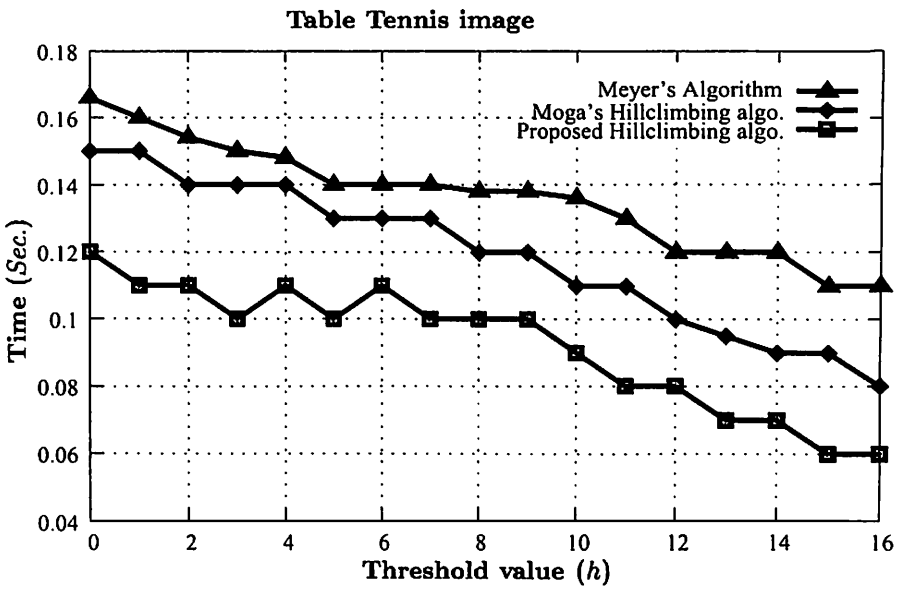


Figure 4.11: Simulation time while running Meyer's ordered queue based algorithm [BM93], Moga's Hillclimbing technique [Mog97] and proposed hillclimbing technique on Tennis image.

identified by both algorithms in an almost identical manner ($P \approx 1$).

Test Image	Threshold value (h)	Number of homogeneous regions	Mismatched pixels	Performance measure (P)
Claire (352×288)	10	17	23	0.99965
Tennis (352×288)	13	22	16	0.999842
Heart (256×256)	8	14	36	0.999451
Akiyo (164×144)	14	22	12	0.999492
MRI Brain (256×256)	12	38	31	0.999527
Cermet (256 × 256)	30	61	37	0.999435
Washinton_IR (250×250)	16	46	61	0.999024
Steel Fissure (290×369)	15	10	18	0.997523
Blood Cells (212×353)	26	20	6	0.99992

Table 4.2: Quantitative evaluation of the segmented images

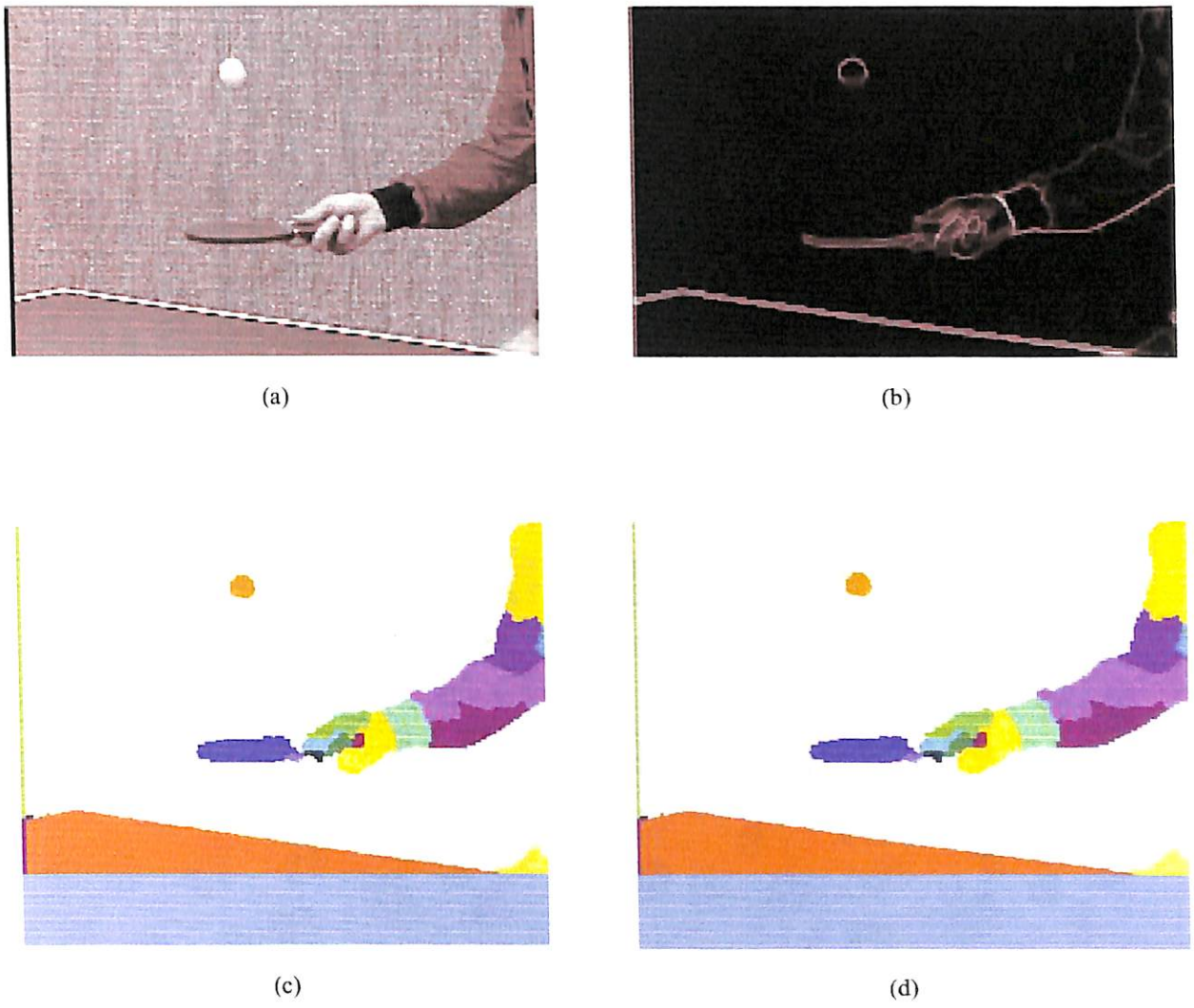


Figure 4.12: (a) Original Tennis image; (b) Morphological multi-scale gradient operation on Table Tennis image with a threshold value of 13; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the proposed watershed algorithm.

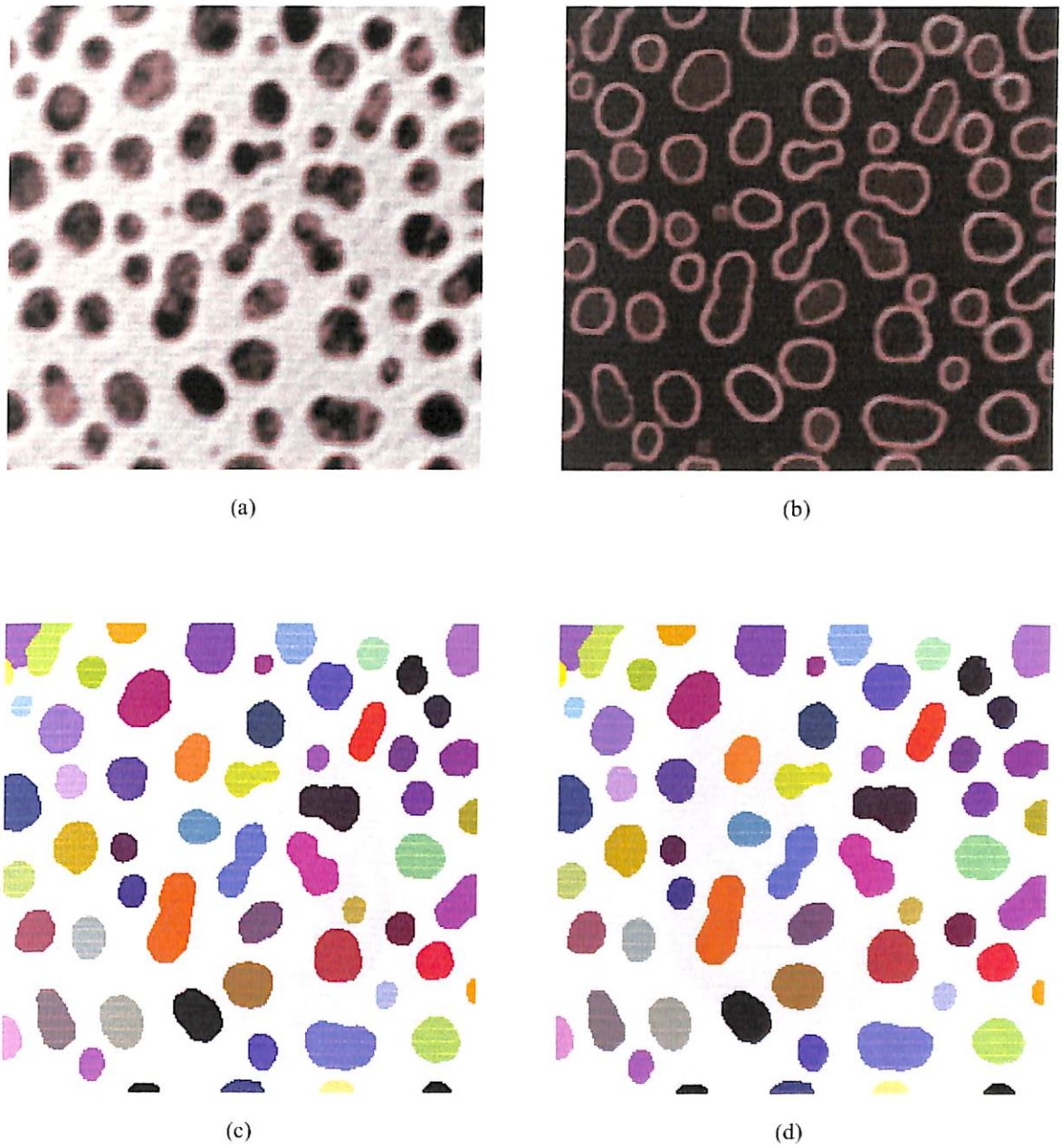


Figure 4.13: (a) Original Cermet image; (b) Morphological multi-scale gradient operation on Cermet image with a threshold value of 30; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the proposed watershed algorithm

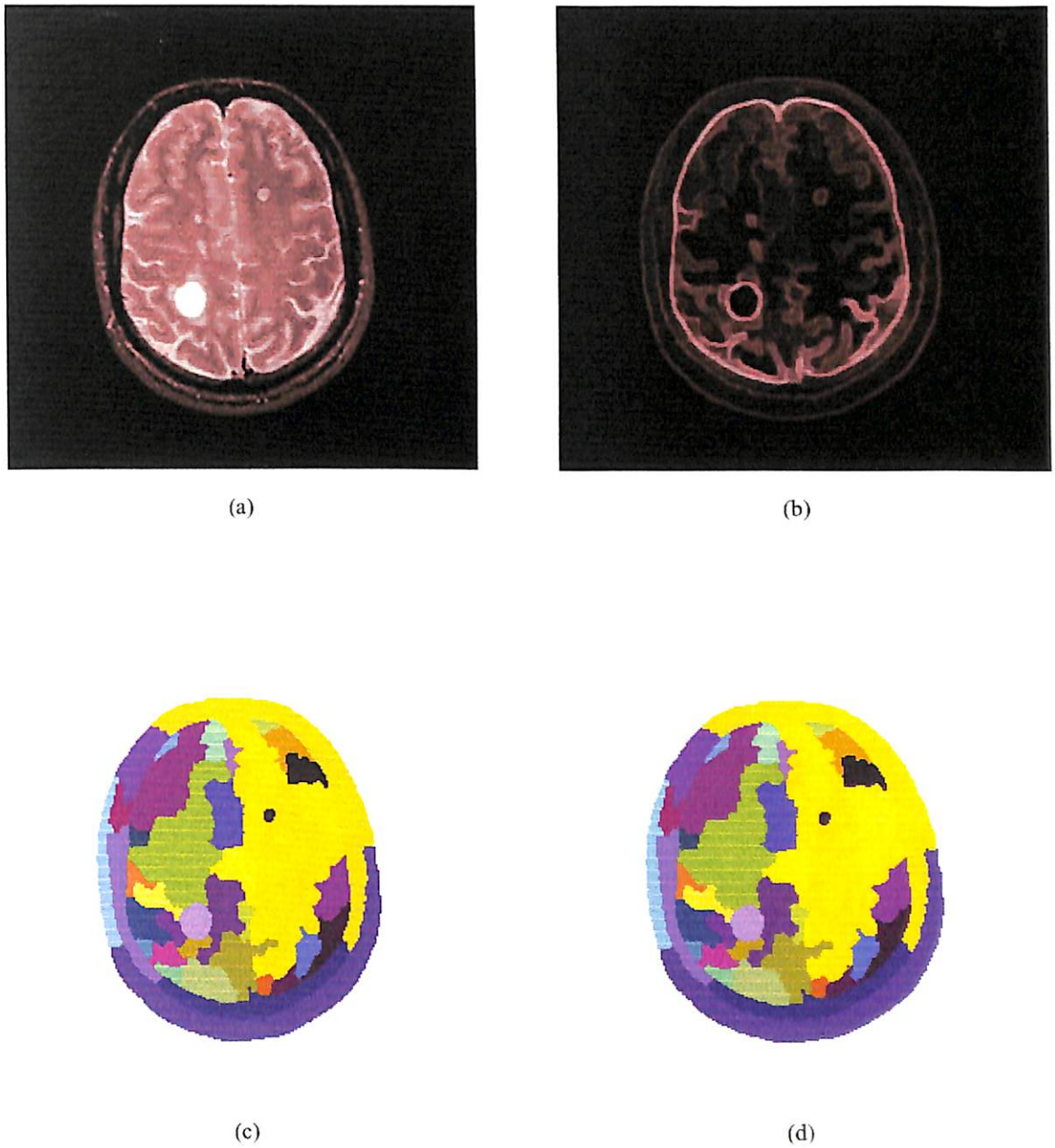


Figure 4.14: (a) Original MRI Brain image; (b) Morphological multi-scale gradient operation on MRI Brain image with a threshold value of 12; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the proposed watershed algorithm.

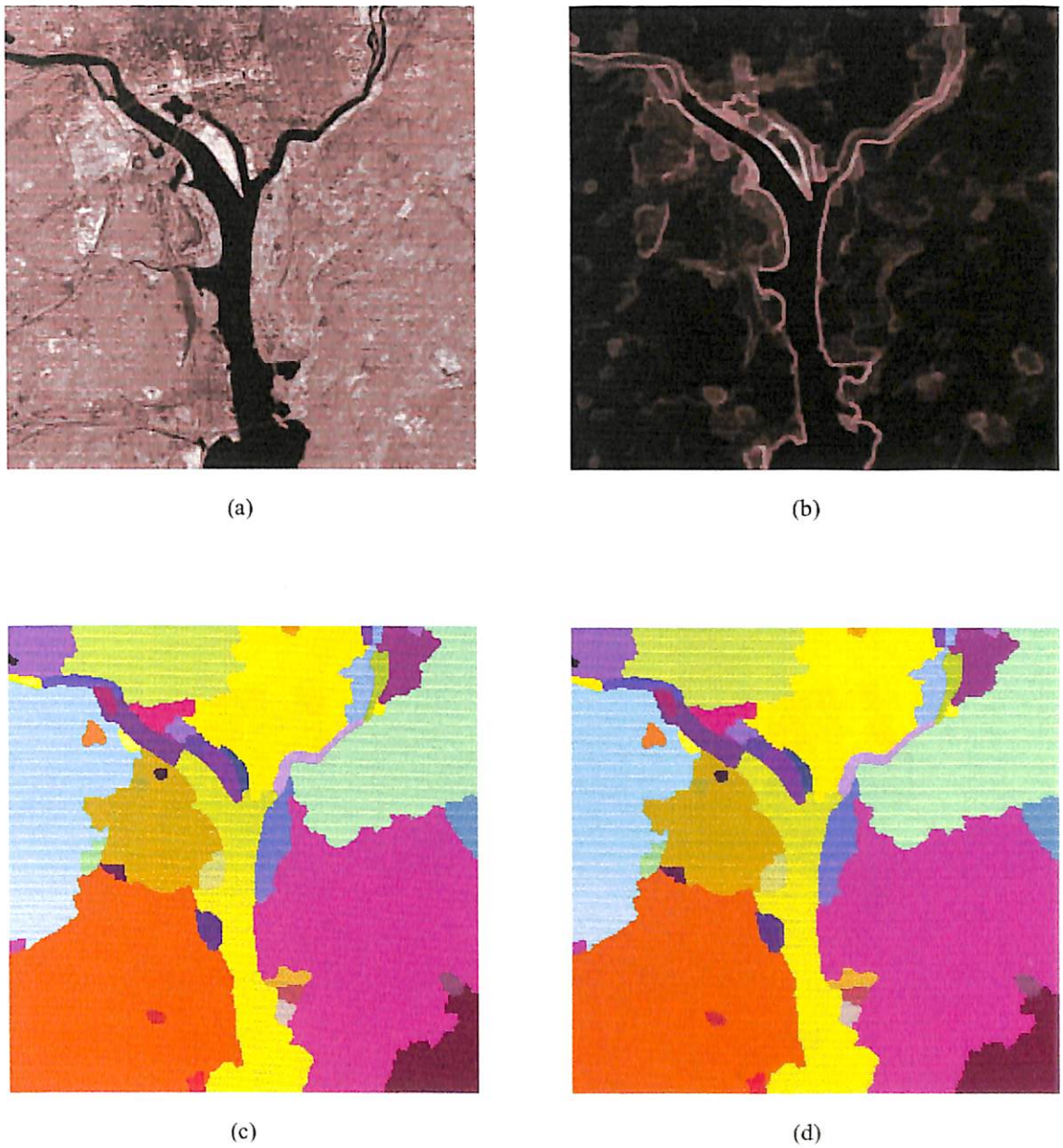


Figure 4.15: (a) Original Washington_ir image; (b) Morphological multi-scale gradient operation on Washington_ir image with a threshold value of 16; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the Proposed watershed algorithm.

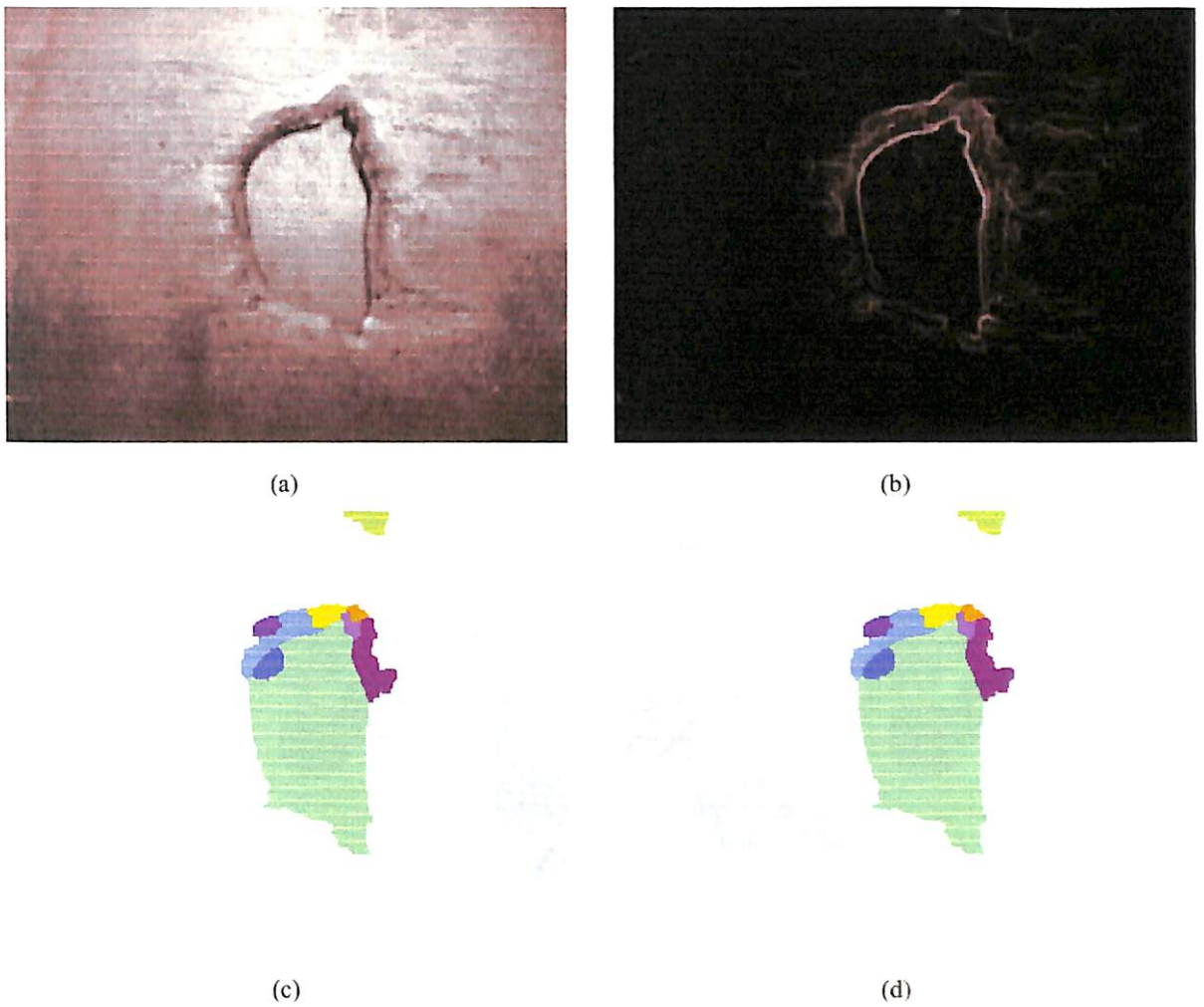


Figure 4.16: (a) Original Steel Fissure image; (b) Morphological multi-scale gradient operation on Steel Fissure image with a threshold value of 15; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the proposed watershed algorithm.

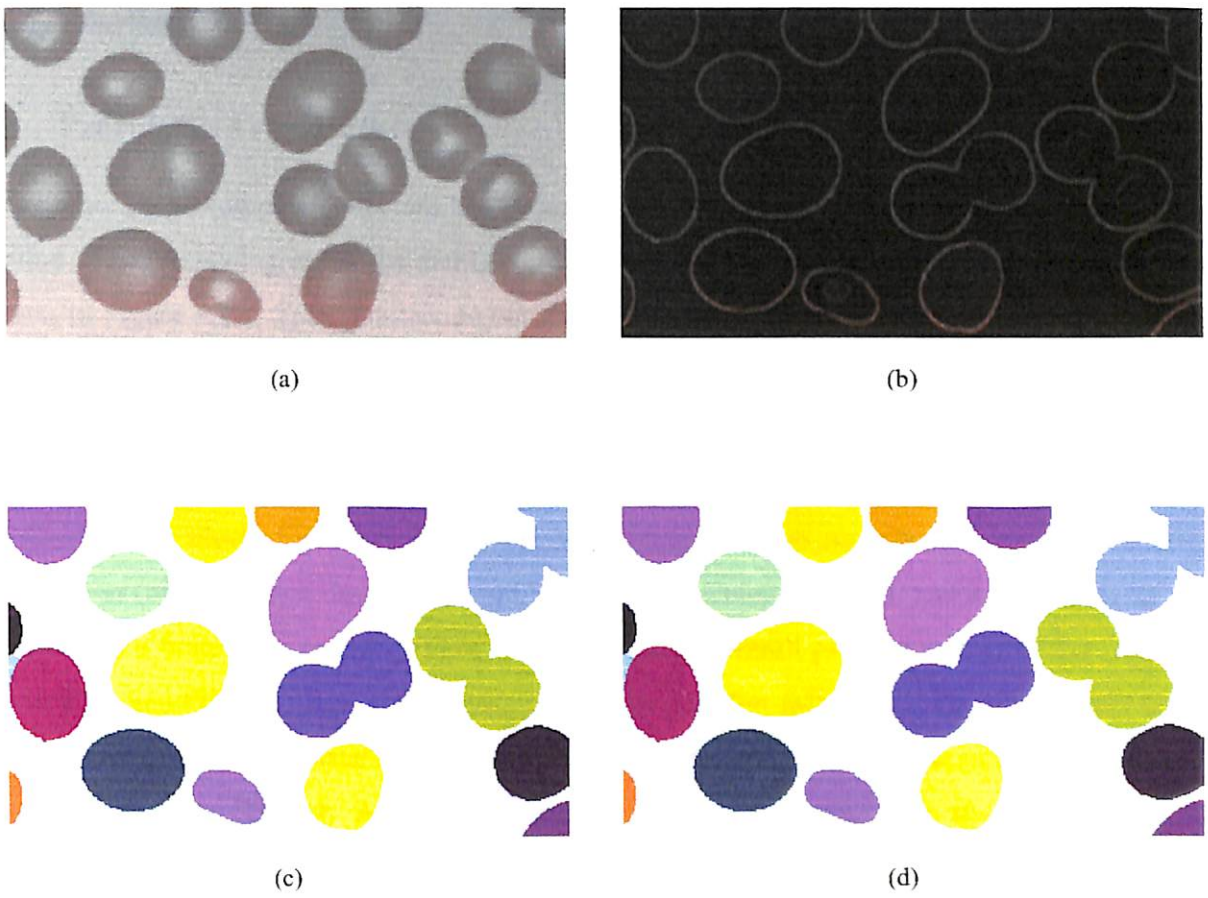


Figure 4.17: (a) Original Blood cell image; (b) Morphological multi-scale gradient operation on Blood cell image with a threshold value of 26; (c) Segmentation produced by Moga's watershed algorithm [Mog97]; (d) Segmentation produced by the proposed watershed algorithm.

4.5 Proposed Prototype Architecture and its FPGA implementation

This section presents a detailed discussion on a prototype architecture for implementing the improved hillclimbing technique proposed in this chapter. Moreover, an FPGA implementation of the architecture is briefly described.

4.5.1 Prototype Architecture

In this section, a prototype architecture has been developed to implement the improved hillclimbing algorithm. The block diagram of the architecture that implements the improved watershed algorithm is shown in Figure 4.18. This architecture consists of four image RAM blocks, which are meant to hold the input image (GRADIENT INPUT IMAGE RAM), the output image of labels (LABEL OUTPUT IMAGE RAM), the steepest neighbor values (STEEPEST LOWER IMAGE RAM) and the lowe: distances (DISTANCE IMAGE RAM), two FIFO Queues, Minima/Non-minima Detection block, Distance Comparison block, Conditional Flooding block, four counters namely Address Counter (up), Label Counter (up), Distance Counter(up) and Wave_Length Counter(down), and a Control Unit. The control unit is a finite state machine (FSM) that controls the overall process including the initialization of RAMs and FIFOs, detection of Plateau of Minima and Non-minima, labeling of Local Minima and Computation of Lower Distance, and finally the Flooding process. Raster scan can be performed by using the Address Counter. A control signal scan_over is generated, when the Address Counter reaches the last pixel address (max_image_size) of image. The Wave_Length Counter is used to assign the proper distance during the analysis of the non-minima plateaus (or Lower Distance Computation process). Initially, it will be loaded with the contents of FIFO_Q2. The Wave_Length Counter is always decremented whenever a pixel is dequeued from FIFO_Q2. When the Wave_Length Counter reache the value 0, a control signal flag_wlc is enabled. Description of the individual blocks follows next.

Neighbor Address Generation Block

The neighborhood addresses are generated in parallel through addition/subtraction of the Image_Width W and the C_address as shown in Figure 4.19(a). Binary addresses are used to generate the addresses of all the neighbors. The width of the image is known to the user and is stored in a register Image_Width If the center pixel C in an image belongs to a boundary, some neighbor addresses may be incorrectly

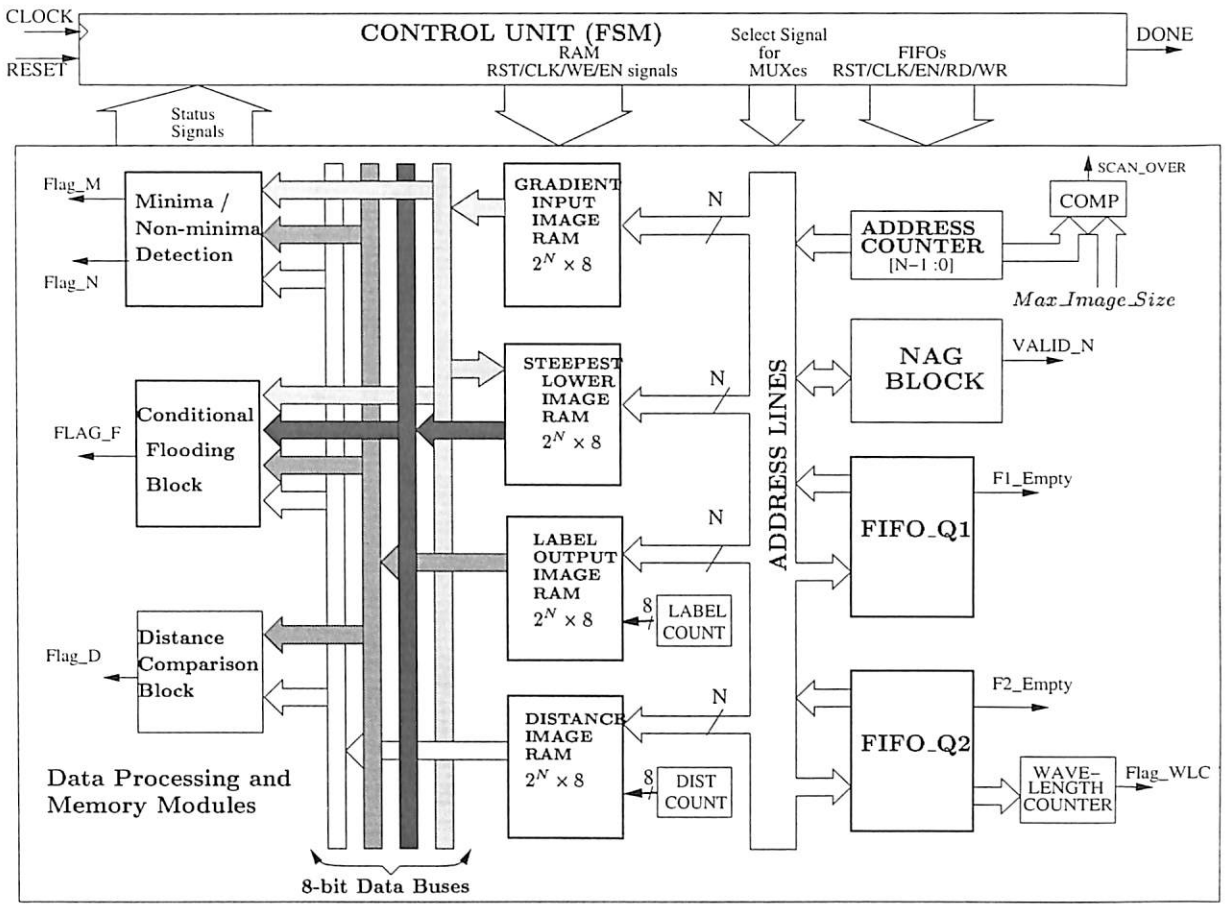


Figure 4.18: Complete architecture of the proposed watershed transform.

computed. Therefore, all these neighbor addresses are compared with the boundary conditions and the corresponding flags are generated, which shows the validity of the neighbors address. Through neighbor selection signal sel_N, any neighbor address can be selected.

The schematic diagram of the circuitry required to correctly generate the address of a pixel N_1 is given in Figure 4.19(b). The circuit blocks labeled ADD and 2's COMP_SUB are used to generate the neighbor pixel address (N_Addr_1) from the center pixel address (C_Address). The block REM is a remainder circuit which is used to find the validity of the generated neighbor address. If the center pixel belongs to a boundary, the neighbor address may be incorrectly computed. To prevent this, the remainder left on dividing by the C_Address and the Image_Width is computed. The remainder determines the validity of the neighboring pixel (whether it is inside or outside the image) as the read_flag is generated.

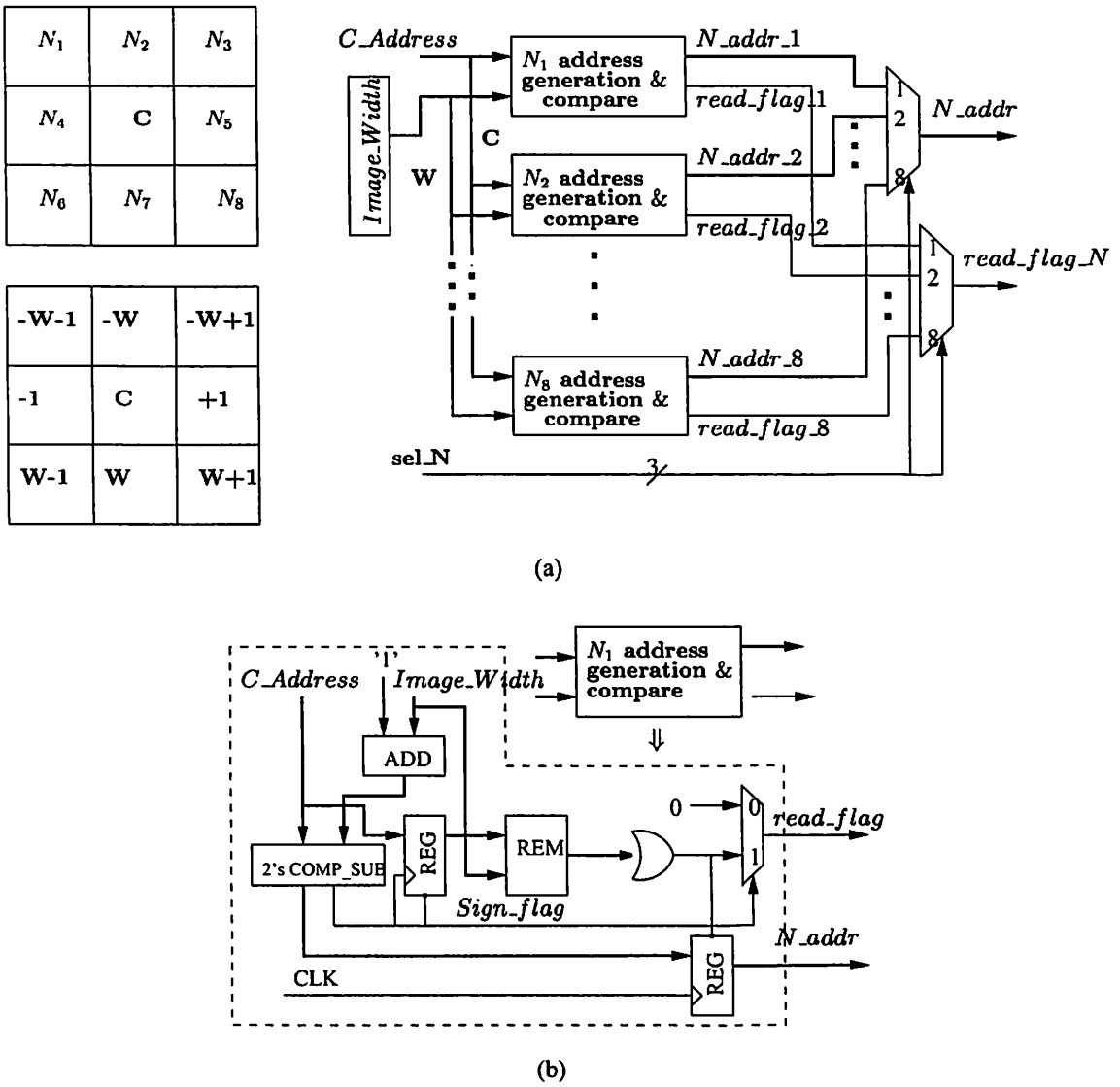


Figure 4.19: (a). The block diagram of the NAG Block. (b). Schematic diagram for generating address of N_1.

Minima/Non-minima Detection

This block diagram as shown in Figure 4.20 is used to detect whether the neighboring pixel belongs to a plateau or is a center pixel belonging to a non-minima plateau. Two control flags are generated on the basis of different conditions. First, the control unit stores the center pixel data of gradient image into a register CREG through the demux. In following clock cycles, the neighbor pixel data are available on the input_data and N_data inputs. The C_dist input always contains the modified center pixel distance. The signal Flag_M is generated when the neighboring pixel data (N_data) of the gradient image and the

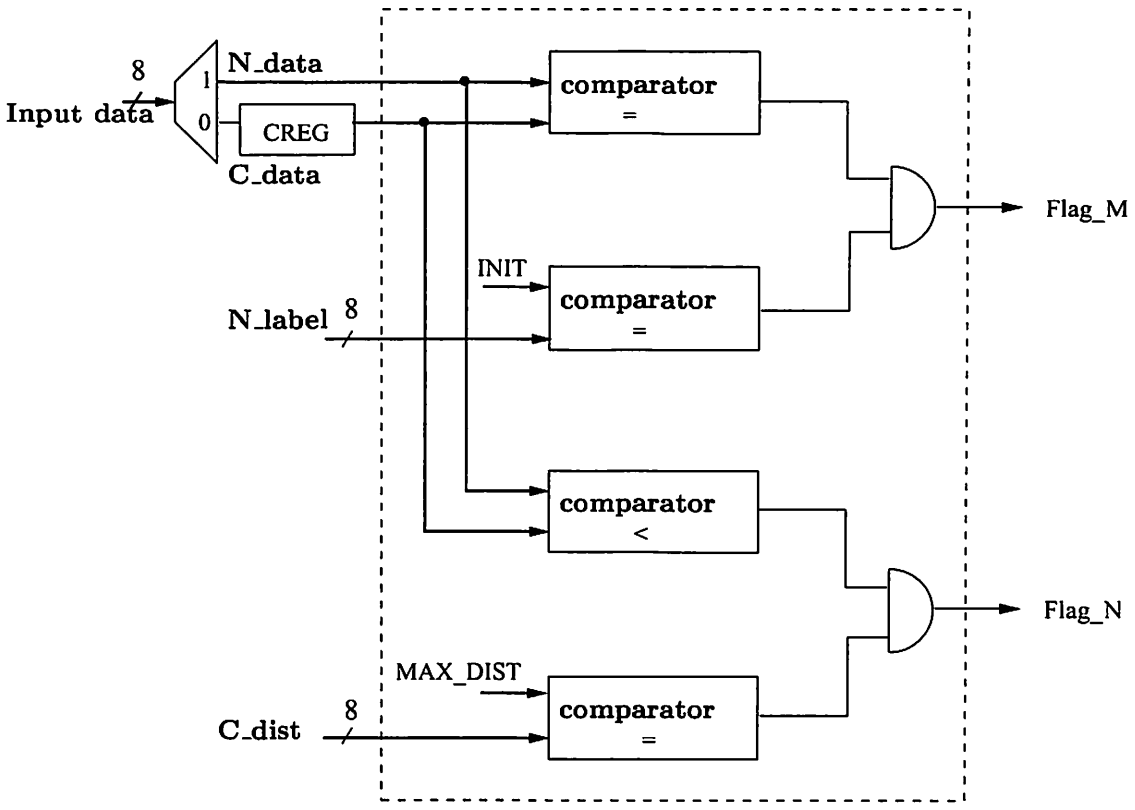


Figure 4.20: Minima/non-minima detection block.

label image (N_label) are equal to center pixel data (C_data) of the gradient image and the initialized value of label image INIT respectively. It indicates that the neighboring pixel belongs to its plateau and not visited previously. This status signal causes the control unit to generate the necessary signal to enqueue the address of the neighbor pixel into FIFO_Q1.

The signal Flag_N is generated when the neighbor pixel data (N_data) of gradient image is less than its center pixel data (C_data), and the center pixel data of distance image is equal to MAX_DIST. It indicates that the center pixel belongs to the non-minima plateau and it is an outer pixel. This status signal causes the control unit to generate the necessary signals to enqueue the address of the outer pixel into FIFO_Q2.

Distance Comparison Block

The distance comparison block shown in Figure 4.21 detects whether the neighboring pixel belongs to a non-minima plateau for which no distance is assigned. The control signal Flag_D is generated when the neighboring pixel label data (N_label) and the distance image data (N_dist) are equal to the

label counter value and the initialized value of the distance image MAX_DIST respectively. It indicates that the neighboring pixel belongs to a non-minima plateau and no distance is assigned to it. This signal makes the control unit generate the necessary signal to enqueue the address of the neighboring pixel into FIFO_Q2 for further exploration and assignment of the current distance value of the WAVE_LENGTH_COUNTER to the distance image pixel.

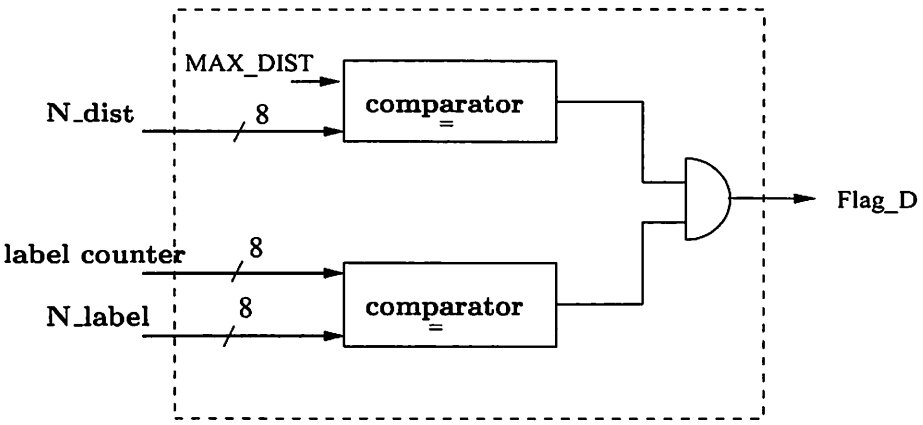


Figure 4.21: Distance comparison block.

Conditional Flooding Block

The block diagram shown in Figure 4.22 detects whether the neighboring pixel has the steepest arc to the center pixel or has lower distance equal to one more than that of the center pixel if both are on a plateau of non-minima. At first, the center pixel distance is assigned to a register DREG through demux. In the next clock cycle, the neighbor pixel data is compared with the center pixel data as N_data with NARM, sl_data with C_data, and dist_data must be 1 or equal to DREG +1. A control signal flag (Flag_F) is generated when the above condition is met, which decides the propagation of the label of the center pixel to its neighbors.

Control Unit (FSM)

The control unit (FSM) generates the reset signal to all the blocks as shown in the complete architecture of Figure 4.23. Moreover, it generates the signals to control the MUXex and the DEMUXes, read, write, enable signals for FIFOs, RAMs and load, enable, reset signal for counters on the basis of the control signals generated through "Detection of Minima/Non-minima" block, "Distance Comparison"

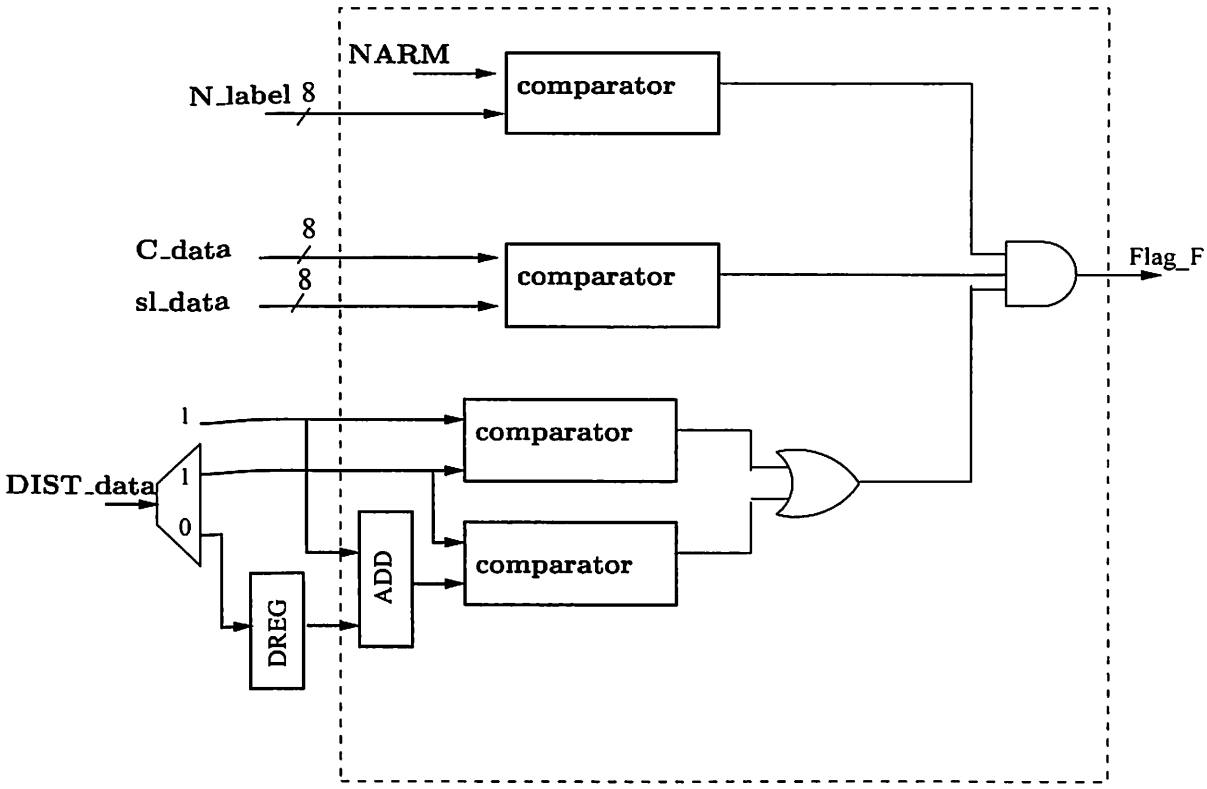


Figure 4.22: Conditional flooding block.

block, "Conditional Flooding" block and the signals generated through FIFOs (f1_empty, f2_empty). After computing the watershed segmentation, a DONE signal is set to 1. The control sequence of this architecture is as follows.

Step 1: Reset the Address Counter, Label Counter, Distance Counter. Initialize all flags to zero. Write DATA_IN into Gradient Input Image RAM block. And initialize Label Output Image RAM block with INIT, and Distance Image RAM block with MAX_DIST.

Step 2: **If** Scan_Over is 1 **then** reset FIFO_Q1 and Address Counter, and go to step 16. **else** increment Address Counter.

Step 3: Read the label Lab_data from Label Output Image RAM addressed by Address Counter. Compare Lab_data with INIT.

if Flag_I is 1 **then** reset FIFO_Q1 and FIFO_Q2 (through signal F1_GSR & F2_GSR) **else** go to step 2.

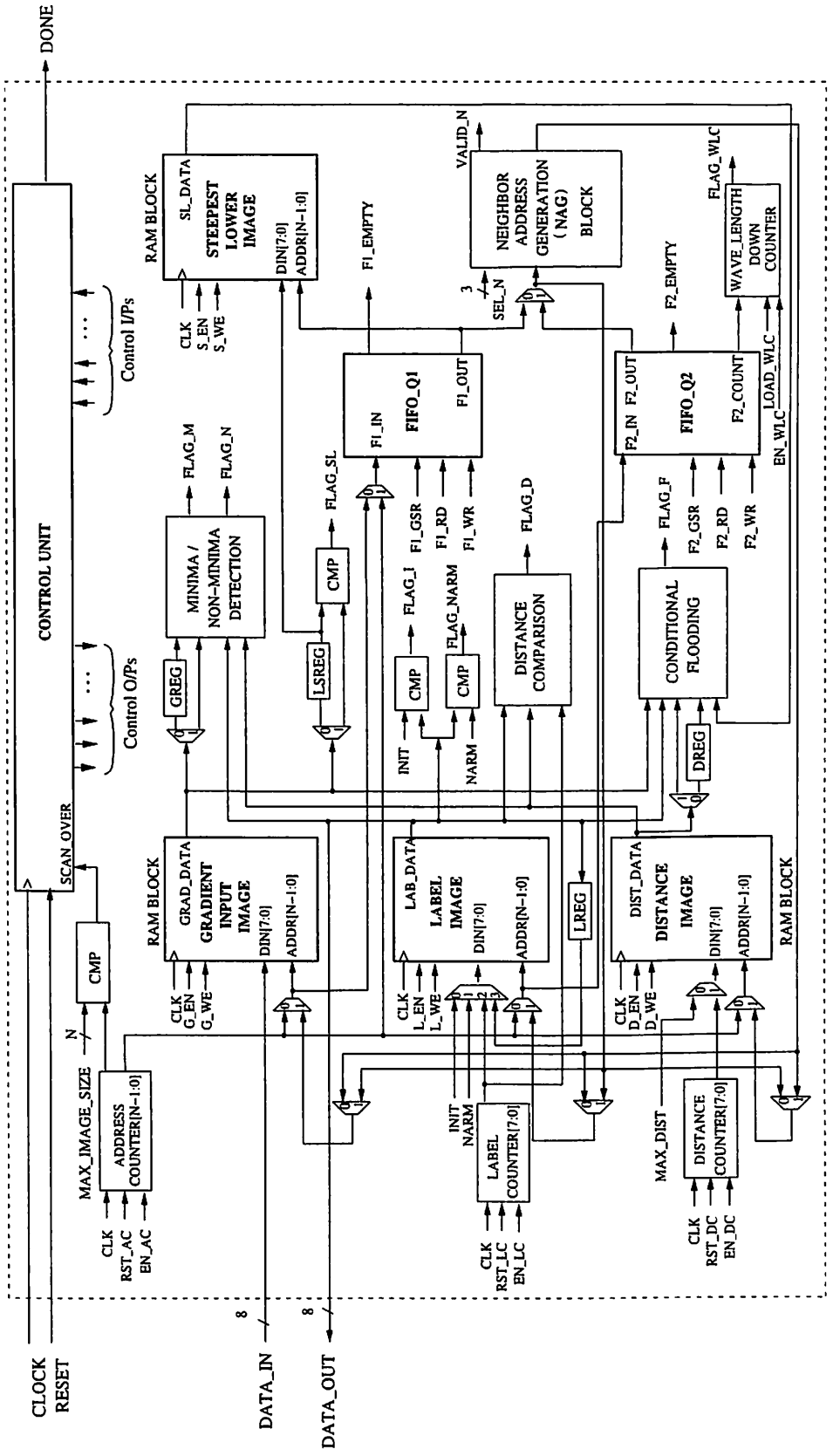


Figure 4.23: Complete architecture of watershed transform.

Step 4: Write the Label Counter value in Label Output Image RAM to address pointed by Address Counter and enqueue this address to FIFO_Q1.

Step 5: **If** F1_empty is 1 **then** go to step 9 **else** dequeue candidate pixel address from FIFI_Q1.

Step 6: Modify the value of the registers GREG and SLREG with Grad_data. Enable NAG block with sel_N = "000".

Step 7: **if** Valid_N is 1 **then** read Grad_data, Lab_data and Dist_data, and transfer the values to Detection of Minima/Non-minima block, which operates as follows.

If Flag_M is 1 **then** write the Label Counter value in Label Output Image RAM to the address pointed by NAG block, and enqueue this address to FIFO_Q1.

If Flag_N is 1 **then** write the Distance Counter value in Distance Image RAM to the address pointed by the NAG block, and enqueue this address to FIFO_Q2.

If Flag_SL is 1 **then** modify SLREG with Grad_data.

Step 8: **If** sel_N = "111" **then** write SLREG in Steepest Lower Image RAM block to address pointed by candidate pixel address **and** go to step 5 **else** increment sel_N and go to step 7.

Step 9: **If** F2_empty is 1 **then** increment Label Counter and go to step 2.

Step 10: Load the Wave-Length Counter with F2_COUNT and increment Distance Counter.

Step 11: **If** F2_empty is 1 **then** reset Label Counter and go to step 2 **else** dequeue the candidate pixel address from FIFO_Q2.

Step 12: **If** Flag_narm is 0 **then** write NARM in Label Output Image RAM to address pointed by FIFO_Q2. Enable NAG block with sel_N = "000".

Step 13: **If** Valid_N is 1 **then** read Dist_data and Lab_data addressed by NAG Block, and transfer the same to the Distance Comparison block, which operates as follows.

If Flag_D is 1 **then** write Distance Counter value in RAM and NARM in Label Output Image RAM to address pointed by NAG block. Also, enqueue this address to FIFO_Q2.

Step 14: **If** sel_N = "111" **then** decrement Wave-Length Counter **else** increment sel_N and go to step 13.

Step 15: **If** Flag_WL is 1 **then** go to step 10 **else** go to step 11.

Step 16: **If** Scan_Over is 1 **then** goto step 18 **else** increment Address Counter.

Step 17: **If** Flag_narm is 0 **then** enqueue Address Counter value to FIFO_Q1. Always goto step 16.

Step 18: **If** F1_empty is 1 **then** goto step 22 **else** dequeue candidate pixel address from FIFO_Q1.

Step 19: Modify the register DREG with Dist_data from the Distance Image RAM and LREG with Label Output Image RAM from to address pointed by FIFO_Q1. and Enable NAG block with sel_N = "000".

Step 20: **If** Valid_N is 1 **then** read Grad_data, Dist_data, Lab_data and SL_data addressed by NAG block and forward them to Flooding Conditional block, the operation of which is described next. **If** Flag_F is 1 **then** write LREG value to Label Output Image RAM to address pointed by NAG block. Enqueue this address to FIFO_Q1.

Step 21: **If** sel_N = "111" **then** go to step 18 **else** increment sel_N and go to step 20.

Step 22: Set DONE signal to 1 and Exit.

4.5.2 FPGA Implementation

The proposed architecture described in section 4.5.1, has been described in VHDL and simulated by Modelsim, and synthesized under the Xilinx ISE4.1i system [xil] targeted to virtex FPGA series device (vide Appendix B). The proposed architecture has been simulated for various test images of size (30×30). An image of size greater than 30 × 30 requires more than one chip for its implementation. Due to limitations of the memory on the virtex device, the FPGA implementation has been limited to an image of size 30 × 30. In this design, the block RAMs are configured with 8-bit data lines and 9-bit address lines. For these address lines a 9-bit up counter is required to generate addresses or ADDRESS COUNTER, and other counter like LABEL COUNTER, DISTANCE COUNTER are configured as 8-bit up counters.

The entire design has been simulated on ModelSim Xilinx 5.5b simulator as shown in Fig. 4.24. In the first clock cycle, a reset signal has been applied on an input pin of FPGA. In the second cycle, the control unit (FSM) will reset all counters, block RAMs and FIFOs. Also a DONE signal is set to 0 by the control unit. From the third cycle onwards, data on DIN pin will be stored in GRADIENT INPUT IMAGE DATA RAM corresponding to the address generated by the Address Counter. A Scan_Over

control signal will become 1 when Address Counter reaches the last address (MAX_IMAGE_SIZE) of the image. The control signal DONE is set to 1 when the entire watershed computation is completed.

Tests were run assuming a clock frequency of 50 MHz. The simulation results obtained for various test images are given in Table 4.3. From the simulation results, the hardware implementation is seen to be faster than the software version by an order of 3. The software results shown in Table 4.3 are achieved using Linux 9.0 environment on a Pentium IV 1.5 GHz processor system.

Test Image (30 × 30)	**Software Time** (*millisecond*)	**Hardware Time**(μ*second*)
Claire	16	30.16
Tennis	16	29.04
Heart	12	28.74
Akiyo	10	28.45
MRI Brain	20	35.42
Cermet	20	34.96

Table 4.3: Comparison between simulation time of hardware implementation and software implementation

The entire design has been simulated, synthesized at gate level and then implemented on the device XCV100-6PQ240. The performance of the FPGA implementation is limited to 49.86 MHz in XCV100-6PQ240. By using sharing of resources such as FIFOs, counters, muxes and demuxes, and by employing a single FSM for the control unit, this design has less hardware requirement compared to the architecture (vide section 3.4) required to implement the improved flooding-based watershed algorithm. The FPGA design summary of the entire architecture is accommodated in Table 4.4. The design utilizes almost 40% of CLBs and 25% of Registers.

4.6 Conclusions

An improved watershed algorithm based on hillclimbing simulation has been introduced in this chapter, which constructs 0-width watershed lines. Results of software simulation show that the proposed improved watershed algorithm has a better performance for same input images compared to the original algorithm due to Moga [Mog97]. Implementation of Moga's algorithm involves considerably more hardware overhead, because it requires a multiplier unit to compute a lower complete image. Moreover, it will take considerable time to compute. The proposed algorithm avoids computation of the lower complete image at the cost of two additional comparisons in the flooding process. To reduce the

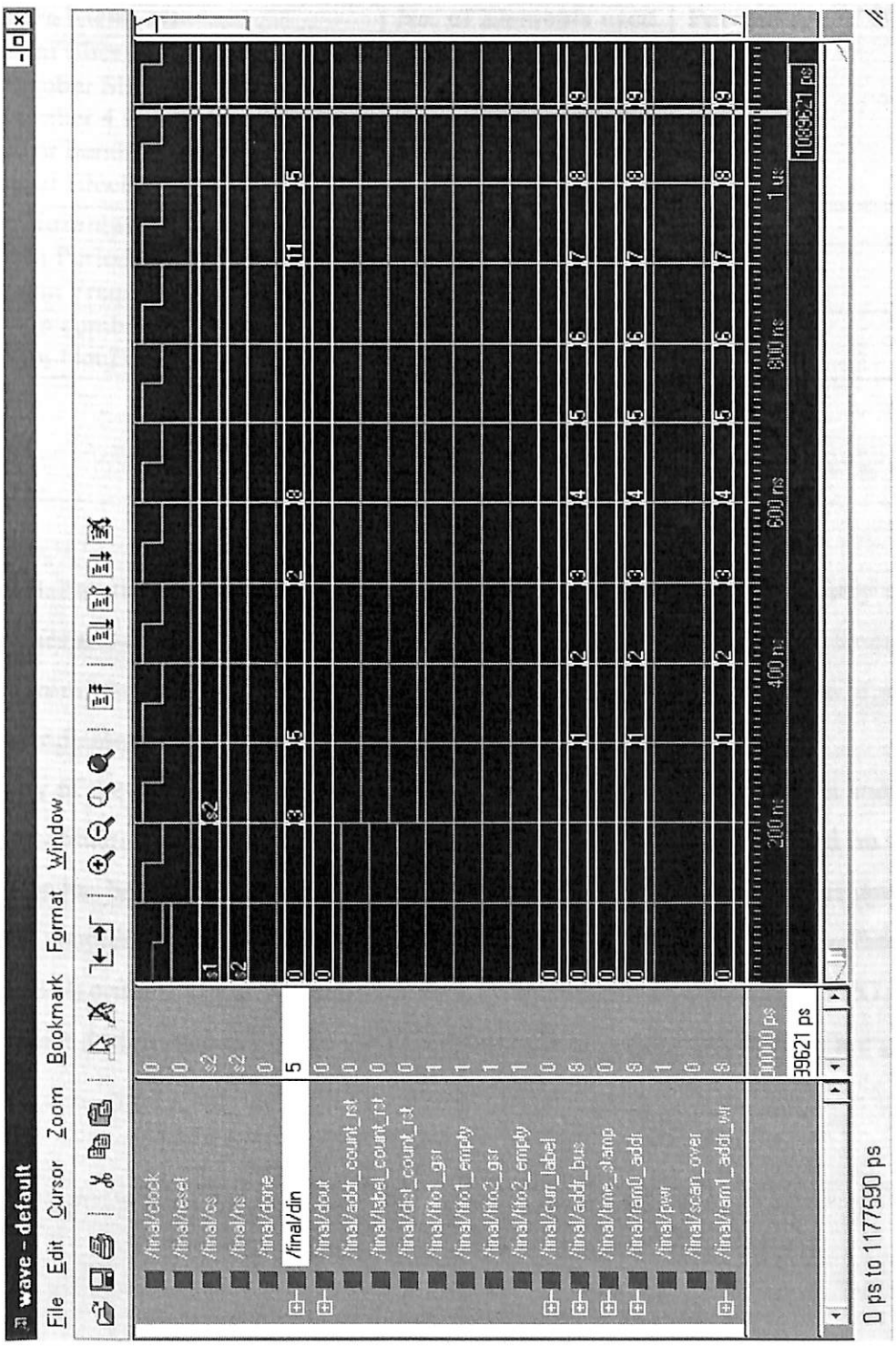


Figure 4.24: Simulation results from ModelSim simulator.

Hardware Elements	No. of Elements used	Percentage of Utilization
Number of Slices	489 out of 1,200	40%
Total Number Slice Registers	261 out of 2,400	25%
Total Number 4 input LUTs	827 out of 2,400	52%
Number of bonded IOBs	18 out of 166	20%
Number of Block RAMs	5 out of 10	50%
Timing Summary:		
Minimum Period:	20.056ns	
(Maximum Frequency:)	49.86MHz	
Maximum combinational Path Delay:	17.152ns	
Maximum Net Delay:	8.65ns	

Table 4.4: Design statistics for the entire architecture.

overall computation time further, some other modification has been adopted, namely employment of two FIFO queues and computation of the steepest lower neighbor image during the process of detection of Minima/Non-minima plateau. Consequently, the reduced complexity makes the algorithm suitable for hardware implementation.

Complexity of the proposed algorithm and the conventional algorithm has been analyzed in detail. Moreover, a quantitative measure of accuracy of the segmentation results produced on various images by the algorithm has been provided. Apart from demonstrating its satisfactory performance on a number of images, this chapter has also described a prototype hardware architecture for an effective realization of the proposed algorithm. The architecture has been synthesized in an appropriate FPGA environment. Also, the relevant design statistics of the FPGA implementation of the architecture are given.

Chapter 5

An Efficient Immersion-Based Watershed Transformation Technique and its Prototype Architecture

This chapter presents an improved watershed algorithm based on the immersion technique and its prototype architecture. Section 5.1 presents a detailed discussion of the conventional immersion algorithm due to Vincent and Soille [VS91]. On highlighting the major features of the improvement, the next section discusses in detail the proposed immersion technique. The computational complexity of the principal steps of both the algorithms is analyzed in section 5.3. Next, section 5.4 presents the simulation results of running the proposed algorithm and the conventional algorithm on different test images for comparison. A 2-D cellular automata (CA) based prototype architecture, which is developed to implement the improved immersion watershed algorithm, and its FPGA implementation results are presented in section 5.5.

5.1 A Conventional Immersion Technique

A sequential watershed transform method based on immersion analogy has been proposed by Vincent and Soillé [VS91], in which the progressive flooding of water on the topographic surface is simulated using a FIFO queue of pixels. Some useful notations are defined next, followed by a detailed description of the immersion-based watershed algorithm [VS91].

5.1.1 Basic Definitions

Let us consider a 2-D digital gray scale image F whose domain is denoted by $D_F \subset \mathbb{Z}^2$ and $F : D_F \longrightarrow \mathbb{N}$ is a function assigning an integer value to each $p \in D_F$. For each pixel $p \in D_F$, $F(p)$ is called the gray value or altitude of p (considering F as a topographical surface).

Geodesic Transformation

Let us now recall the definition of the geodesic distance [LM84], in which A is a set of connected components.

Definition 5.1 (Geodesic distance) *The geodesic distance $d_A(p, q)$ between two pixels p and q in A is the length of the shortest of all the paths between p and q and is totally included in A, that is*

$$d_A(p, q) = \mathbf{inf}\{l(P)\} \ ,$$

in which P *is the path between p and q which is totally included in A, and $l(p)$ represents the length of the path.*

This definition is illustrated in Figure 5.1.

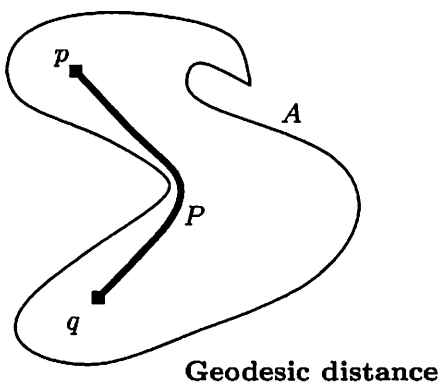


Figure 5.1: The geodesic distance between p and q inside A.

Let $B \subseteq A$ be partitioned into k connected components $B_1, B_2, \ldots, B_k$.

Definition 5.2 (Geodesic influence zone) *The* geodesic influence zone $iz_A(B_i)$ *of a connected component* B_i *of* B *in* A *is the locus of the points of* A *whose geodesic distance to* B_i *is smaller than their geodesic distance to any other component* B_j.

$$iz_A(B_i) = \{p \in A | \forall j \in [1,k]/\{i\}, d_A(p, B_i) < d_A(p, B_j)\}$$

This concept is illustrated in Figure 5.2.

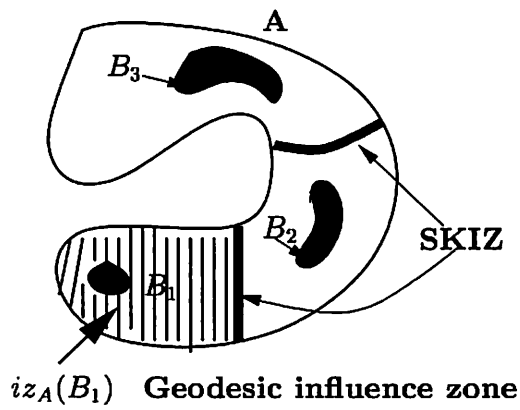


Figure 5.2: The geodesic influence zone and the skeleton by influence zone.

Definition 5.3 (Skeleton by influence zones) *Those points of* A *which belong to the boundary between geodesic influence zones constitute the* skeleton by influence zones *(SKIZ) of* B *inside* A, *denoted by* $SKIZ_A(B)$:

$$SKIZ_A(B) = A/IZ_A(B) \text{ with } IZ_A(B) = \bigcup_{i}^{k} iz_A(B_i)$$

where the set $IZ_A(B)$ is the union of the geodesic influence zones of the connected components of B. The complement of the set $IZ_A(B)$ within A is called the $SKIZ$ (*skeleton by influence zones*).

The algorithmic definition of the watershed by simulated immersion as proposed by Vincent and Soille [VS91] is described next.

Let F: $D_F \rightarrow \mathbb{N}$ be a digital gray-scale image, having h_{min} and h_{max} as the smallest and the largest gray values respectively in its domain D_F. Next, the threshold $T_h(F)$ of F at level h is defined as

$$T_h(F) = \{p \in D_F, F(p) \leq h\}.$$

Let $C(M)$ denote the catchment basin related to a minimum M, and $C_h(M)$ the subset of $C(M)$ such that $C_h(M)$ consists of the points with an altitude smaller than or equal to h, that is,

$$C_h(M) = \{p \in C(M), F(p) \leq h\} = C(M) \cap T_h(F)$$

Moreover, let $MIN_h(F)$ denote the set of points of F belonging to the minima at altitude h.

Definition 5.4 (Watershed by Immersion) *Define the following recursion:*

$$\begin{aligned} C_{h_{min}}(F) &= \{p \in D | F(p) = h_{min}\} = T_{h_{min}}(F) \\ C_{h+1}(F) &= MIN_{h+1}(F) \cup IZ_{T_{h+1}(F)}(C_h(F)), \ \forall h \in [h_{min}, h_{max} - 1] \end{aligned} \quad (5.1.1)$$

The watershed Wshed(F) of F corresponds to the complement of $C_{h_{max}}(F)$ in D_F:

$$Wshed(F) = D_F \backslash C_{h_{max}}(F)$$

That is, the watershed of F comprises the set of all the points of D_F which do not belong to any catchment basins of F. The watershed by immersion concept is illustrated in Figure 5.3.

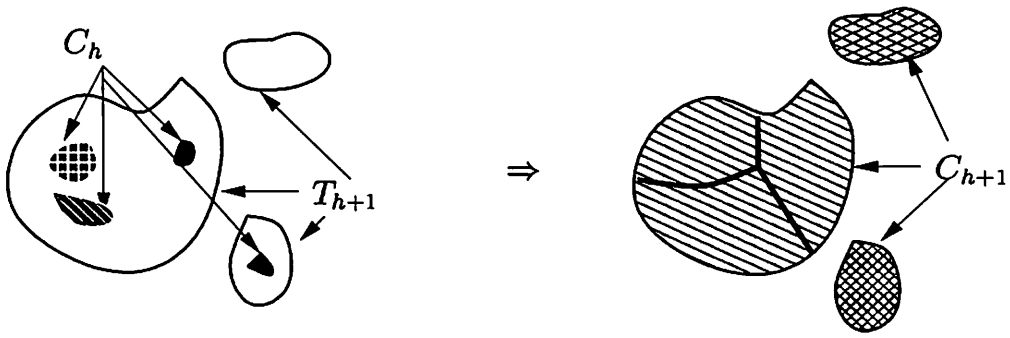


Figure 5.3: Watershed by immersion

The original immersion-based watershed algorithm examines all the pixels in an image at successive gray levels, and operates in two stages, namely a sorting step followed by a flooding step. The sorting step produces a list of the pixels sorted in the ascending order of grayscale values. Once the pixels have been sorted, it is possible to operate on them at successive gray levels, as if a flood were progressing up through the levels of the image. Each minimum is given a unique label. Utilizing a First-In-First-Out (FIFO) data structure, the pixels at altitude $h + 1$ are processed after processing those at altitude h, starting from the lowest altitude. All pixels at the current gray level, that have at least one neighbor at

a lower level, are put in a FIFO queue and assigned the label of the neighbor(s) with the lowest gray level; in addition, its associated distance is given a value equal to one greater than that of its immediate neighbor with lower value. The FIFO queue is then processed. Each pixel p is dequeued from the queue and its non-labeled neighbors $q \in N_G(p)$ at the current height level are put at the end of the queue and assigned the label of p. This allows the minima to spread or flood within the confines of the current height level. At each gray level, pixels are appended to the catchment basins located in their immediate neighborhood at the lower gray levels. Whenever two floods originating from different catchment basins reach each other, a dam is built to prevent the basins from merging. The pixels which are detected to be on the boundaries and are equidistant from two catchment basins, get a special label, namely that of a watershed pixel. A final scan of the gray level identifies and labels the new minima. The simulated immersion process in a sample image F is illustrated in Figure 5.4.

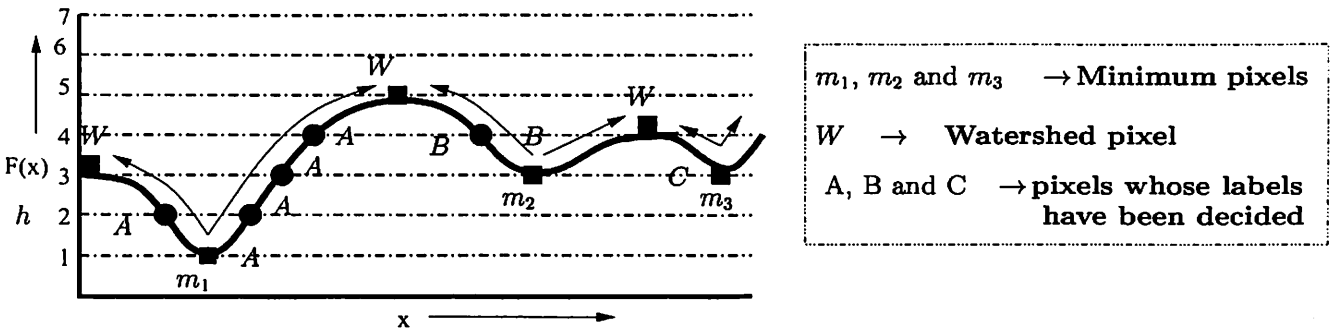


Figure 5.4: Simulated immersion process in a sample image F.

The psuedo-code of the simulated immersion algorithm due to Vincent and Soille [VS91] is given below.

1: **Procedure 5.1** *Watershed_by_Immersion(F, L, dist)*
2: **Input**: Gradient image F. $\{F : D_F \rightarrow \mathbb{N}\}$
3: **Output**: Label image L and distance image $dist$.
4: INIT $\leftarrow -2$; /* Initial value of label image */
5: MASK $\leftarrow -1$; /* initial value at each level */
6: WSHED $\leftarrow 0$; /* label of the watershed pixel */
7: $current\_label \leftarrow 0$; {/* Initialize the Current label */ }
8: FIFO_INIT(Q); /* Initialize the FIFO Queue (Q) */

```
for all p ∈ D_F do
    L(p) ← INIT;   dist(p) ← 0;
end for
SORT pixels in increasing order of gray vales (minimum h_min, maximum h_max).
/* Start Flooding */
for h = h_min to h_max do
    for all p ∈ D_F with F(p) = h {/* mask all the pixels at level h */ } do
        L(p) ← MASK;
        if q ∈ N_G(p) with (L(p) > 0) or (L(q) = WSHED) {/* Initialize queue with neighbors at level
        h of current basins or watersheds */} then
            dist(p) ← 1;   FIFO_ADD(p, Q);
        end if
    end for
    current_distance ← 1;
    FIFO_ADD(FICTITIOUS, Q);
    loop
        p ← FIFO_REMOVE(Q);
        if p = FICTITIOUS then
            if FIFO queue Q is empty then
                BREAK;
            else
                FIFO_ADD(FICTITIOUS, Q);
                current_distance := current_distance + 1;
                p ← FIFO_REMOVE(Q);
            end if
        end if
        Decide_Label_P(p, L, dist, Q, current_distance);   /* procedure for deciding the label of
        the current pixel p */
    end loop
    New_Minima_Detection(F, L, dist, current_label, h);   /* procedure for detecting and pro-
    cessing the new minima at level h */
end for
```

38: /* End of the procedure Watershed_by_Immersion */

Procedure Decide_Label_P used in the procedure Watershed_by_Immersion may be given in pseudo-code form as follows.

1: **Procedure 5.1.1** *Decide_Label_P(p, L, dist, Q, current_distance) /* procedure for deciding the label of the current pixel p */*
2: Input: p, L, $dist$, queue pointer Q, and $current\_distance$.
3: Output: L and $dist$.
4: **for all** $q \in N_G(p)$ {/* labeling p by inspecting neighbors */} **do**
5: **if** ($dist(q) < current\_distance$) and (($L(q) > 0$) or ($L(q)$ = WSHED)) {/* q belongs to an existing catchment basin or to watersheds} **then**
6: **if** $L(q) > 0$ **then**
7: **if** $L(p)$ = MASK or $L(p)$ = WSHED **then**
8: $L(p) \leftarrow L(q)$;
9: **else if** $L(p) \neq L(q)$ **then**
10: $L(p) \leftarrow$ WSHED;
11: **end if**
12: **else if** $L(p)$ = MASK **then**
13: $L(p) \leftarrow$ WSHED;
14: **end if**
15: **else if** $L(q)$ = MASK and $dist(q) = 0$ {/* q is plateau pixel */} **then**
16: $dist(q) \leftarrow current\_distance + 1$;
17: FIFO_ADD(p, Q);
18: **end if**
19: **end for**
20: /* End of the procedure Decide_label_p */

Procedure New_Minima_Detection used in the procedure Watershed_by_Immersion may be described with the help of pseudo-code as follows.

1: **Procedure 5.1.2** *New_Minima_Detection(F, L, dist, current_label, h) /* procedure for detecting and processing the new minima at level h */*
2: **for all** $p \in D_F$ with $F(p) = h$ {/* rescan all the pixels at level h */ } **do**

```
3:    dist(p) ← 0;    {/* Reset distance to zero */ }
4:    if L(p) = MASK {/* p is inside a new minimum */} then
5:       current_label := current_label + 1;
6:       FIFO_ADD(p, Q);
7:       L(p) ← current_label;
8:       while FIFO queue Q is not empty do
9:          q ← FIFO_REMOVE(Q);
10:         for all r ∈ N_G(q) {/* inspect neighbors of q */} do
11:            if L(r) = MASK then
12:               FIFO_ADD(r, Q);
13:               L(r) ← current_label;
14:            end if
15:         end for
16:      end while
17:   end if
18: end for
19: /* End of the procedure New_Minima_Detection */
```

Vincent's immersion-based watershed algorithm [VS91] generally produces incomplete watershed lines and isolated minima. It also suffers from a second drawback, namely a large flat region acquires the watershed labels and the watershed lines will thicken out and spread to fill the plateau between minima resulting in patches of watershed pixels remaining on the segmented output image. A brief explanation of the drawbacks follows.

The first drawback relates to generation of incomplete watershed lines. A pixel which is adjacent to two different catchment basins, and initially gets labeled as a watershed pixel, is allowed to be over-written at the current gray level by the label of another neighboring pixel. This drawback is illustrated in Figure 5.5.

The second drawback is concerned with production of thick watershed lines. The algorithm tries to classify the pixels as watershed pixels at the current gray level. Next, watershed labels have to be propagated, because it may be the case that pixels with gray level h only have watershed pixels in their neighborhood. This drawback is illustrated in Figure 5.6.

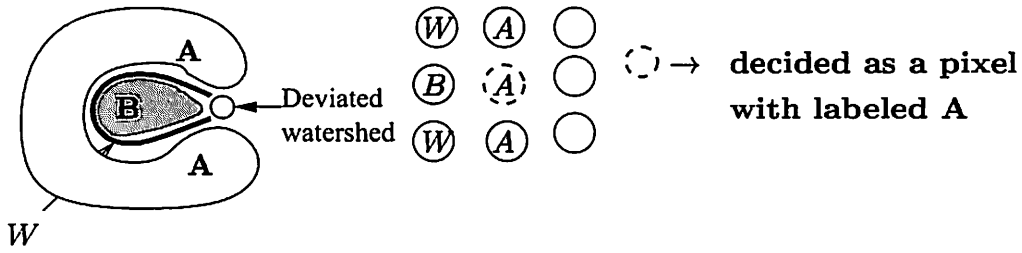


Figure 5.5: Discontinuity in watershed lines.

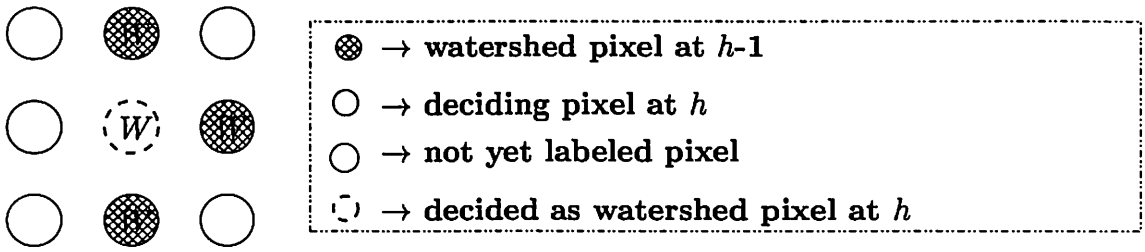


Figure 5.6: Thick watershed line.

The proposed algorithm, which is described in depth in the next section, attempts to eliminate the shortcoming of the conventional algorithm.

5.2 Proposed Algorithm

An improved immersion-based watershed algorithm is described in the present section. It consists of two stages, namely detection and labeling of the local minima, and simulated immersion (flooding). The algorithm starts by detecting and labeling the initial seeds that is, the minima of the gradient, which characterize the regions of interest. The latter are defined as connected plateaus of pixels (plateau of minima) in the gradient image which do not have neighboring pixels of lower gray level. Flooding of the catchment basins (labeling of non-labeled pixels) next starts from the pre-determined regional minima at the lowest value of h and proceeds in the increasing order of h. The flooding process is concerned with deciding the labels of the neighboring pixels with higher or equal altitude based on conditional neighborhood comparisons (involving 16 additional pixels) and the geodesic distance (lower distance), which is the distance from the nearest outer pixel of the plateau. Utilizing a First-In-

First-Out (FIFO) data structure, the pixels at an altitude $h+1$ are processed after processing those at altitude h. All the labeled pixels at the current level h, that have a non-labeled neighboring pixel, are put in a First-in-First-out (FIFO) queue. The queue is then processed. The pixels are dequeued one at a time and subsequently the labels of the non-labeled neighboring pixels at a higher or equal altitude are determined. The processing of the neighboring pixels at a higher or equal altitude reduces the problem to calculating the geodesic *skeleton by influence zone* (SKIZ) [BL79]. Thus, the non-labeled pixels are assimilated into different components in an increasing order of gray levels. Inside the flat areas, which are not yet labeled and called plateaus of non-minima, components progress synchronously, so that they incorporate equal extents within the plateau. Consequently, a pixel on such a plateau is labeled along the shortest path completely included in the plateau to a lower downward brim of the plateau.

Based on the above description and the definitions given in section 5.1.1, the construction of watershed by modified immersion is defined as follows.

Definition 5.5 (Watershed by modified immersion)

$$\begin{aligned} C_{h_{min}}(F) &= MIN_{h_{min}}(F) = \{p \in D_F \mid F(p) = h_{min} \textit{ such that } \exists q \in N_G(p),\ \mathrm{F}(q) \geq \mathrm{F}(p)\} \\ \mathrm{C}_{h+1}(F) &= MIN_h(F) \cup IZ_{T_{h+1}(F)}(\mathrm{C}_h(F)),\ \forall h \in [h_{min}, h_{max}-1] \end{aligned}$$

with the zones of influence defined as

$$\begin{aligned} iz_{T_{h+1}(F)}(C_h(i)) &= \{p \in T_{h+1}(F), \forall j \in [1,k]/\{i\}, d_{T_{h+1}(F)}(p, C_h(i)) \leq d_{T_{h+1}(F)}(p, C_h(j)\} \\ IZ_{T_{h+1}(F)}(\mathrm{C}_h(F)) &= \bigcup_{i \in [1:k]} iz_{T_{h+1}(F)}(C_h(i)) \end{aligned}$$

The watershed WSHED (F) of F is the complement of $C_{h_{max}}(F)$ in D_F:

$$WSHED(F) = D_F \setminus C_{h_{max}}(F)$$

where $\setminus$ represents the set difference operation.

In the proposed algorithm, flooding of the catchment basins starts from the pre-determined regional minima and the conditional neighborhood comparisons while processing the eight non-labeled neighboring pixels of a labeled center pixel. As a result, always a continuous and thin watershed line can be found quickly. The proposed algorithm shows better computational complexity and throughput than the algorithm [VS91] due to Vincent and Soille as it is able to decide labels of eight pixels at a time in contrast to the algorithm [VS91], which does so for a single pixel. Moreover, the proposed

algorithm produces watershed lines possessing a uniform width of a single pixel in contrast to the algorithm [VS91] that may give rise to thick watershed lines. The proposed algorithm also derives its strength from the parallelization of the label propagation process involved in the flooding procedure.

5.2.1 Local Minima Detection and Labeling

In this procedure, a single raster scan has been employed, and two FIFO queues Q_{PD} (for plateau detection) and Q_{PA} (for plateau analysis) have been utilized. For each not yet labeled pixel p, its 4-connected or 8-connected neighborhood is inspected. Thus, if all the neighbors are of higher gray level than p, then p is deemed to be an *isolated minimum*, and a label is assigned to it. Else, if this pixel belongs to a plateau, the plateau is scanned in a breadth-first order, and the visited pixels are correctly labeled with the current label, and inserted into the queue Q_{PD} used for detection of plateau. The examination always starts from an *inner* pixel which introduces the neighboring pixels of equal altitude in the list of the candidates. A currently investigated candidate pixel may still qualify as an *inner* pixel; otherwise, it is an *outer* pixel. The lower distance of the outer pixel is 1, and inserted into the queue Q_{PA} for analysis of the plateau. After scanning the plateau, if the queue Q_{PA} is found to be not empty (indicating that a non-minima plateau has been detected), it is then necessary to compute the distance of the *inner* pixels (from the *outer* pixels) and assign a NARM (Not A Regional Minimum) label to the plateau under consideration for subsequent flooding. The pseudo-code of the whole procedure of detection and labeling of local minima is given below.

Procedure 5.2 *LM_Det_Labeling(F, L, d) /* Procedure for detecting and labeling local minima */*
Input: Morphological multi-scale gradient image F. $\{F : D_F \rightarrow \mathbb{N}\}$
Output: Label image L and distance image d. $\{L, d : D_F \rightarrow \mathbb{N}\}$
Let G be a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$ and $N_G(p)$ stand for the neighbors of a pixel p on the grid G.
INIT $\leftarrow -2$;{/* initialization value */}
NARM $\leftarrow -1$;{/* Not A Regional Minima */}
MAX_DIST $\leftarrow -1$;
$current\_label \leftarrow 0$ {/* initialize the Current label */ }
for all $(p \in D_F)$ **do**
    $L(p) \leftarrow$ INIT; $d(p) \leftarrow$ MAX_DIST; /* label image and distance image initialization */
end for

12: **for all** ($p \in D_F$ with $L(p)$ = INIT) {/* for each pixel in the image which is not yet visited during the raster scan procedure */} **do**
13: Neighborhood_Inquiry(p, F, *pixel_type*); /* procedure for inquiring the neighborhood pixels of p */
14: **if** *pixel_type* = MINIMUM {/* it is an *isolated minimum* */} **then**
15: $L(p) \leftarrow$ *current_label*++;
16: **else if** *pixel_type* = $ON\_PLATEAU$ {/* a plateau is detected */} **then**
17: Plateau_Analysis(p, F, L, d, *current_label*); /* procedure for plateau detection and its labeling */
18: **else if** *pixel_type* = NON_MINIMUM {/* a non-minimum pixel is detected */} **then**
19: $L(p) \leftarrow NARM$;
20: **end if**
21: **end for**
22: /* End of procedure LM_Det_Labeling */

Procedure Plateau_Analysis used in the procedure LM_Det_Labeling may be described with the help of pseudo-code as follows.

1: **Procedure 5.2.1** *Plateau_Analysis(p, F, L, d, current_label) /* procedure for plateau detection and its labeling */*
2: Input: p, F, L, d, *current_label*.
3: Output: L, *current_label*.
4: $L(p) \leftarrow$ *current_label*;
5: FIFO_INIT(Q_{PD}); FIFO_INIT(Q_{PA});{/* Initialize the FIFO Queue (Q_{PD}) and (Q_{PA})*/}
6: FIFO_ADD(p, Q_{PD}); {/* insert the pixel p into the queue Q_{PD} */}
7: **while** fifo queue Q_{PD} not empty {/* plateau Detection phase */} **do**
8: $p \leftarrow$ FIFO_REMOVE(Q_{PD}); {/* get candidate pixel p from queue Q_{PD} */}
9: **for all** $q \in N_G(p)$ {/* inquire neighboring pixels of p */} **do**
10: **if** (($L(q)$ = INIT) and ($F(p) = F(q)$)) **then**
11: $L(q) \leftarrow$ *current_label*;
12: FIFO_ADD(q, Q_{PD});
13: **else if** ($F(q) < F(p)$ and $d(p)$ = MAX_DIST) **then**
14: $d(p) \leftarrow 1$;

```
15:          FIFO_ADD(q, Q_PA);
16:       end if
17:    end for
18: end while
19: if FIFO QUEUE Q_PA is empty then
20:    current_label := current_label +1;
21: else
22:    Compute_Lower_Distance(L, d, Q_PA);
23: end if
24: /* End of procedure Plateau_Analysis */
```

Procedures Neighborhood_Inquiry and Compute_Lower_Distance, which are used in LM_det_Labeling have been described as procedures 3.3.1 and 4.3.1.1 in chapters 3 and 4 respectively.

5.2.2 Simulated immersion

In the proposed scheme, the label propagation of the catchment basins starts from their regional minima according to the gray level h ($h_{min} \leq h \leq h_{max}$). The label propagation, which decides the labels of the neighboring ($N_1 - N_8$) pixels with equal or higher altitude than the center pixel C and requires 16 additional supporting pixels ($S_1 - S_{16}$), is illustrated in Figure 5.7. Utilizing a single FIFO queue Q_I, the pixels at gray level h+1 are processed after processing those at gray level h. Initialize the FIFO queue Q_I with the labeled pixels at level h which have at least one NARM or WRD (watershed) neighboring pixel with equal or higher altitude. Pixels are dequeued from the queue Q_I one at a time and the labels of NARM or WRD neighboring pixels ($N_1 - N_8$) are decided next, based on the conditional neighborhood comparisons. If the neighboring pixels have the same altitude as the center pixel C, then the lower distance (vide definition 3.7, Chapter 3) needs to be made use of to decide their label. Only those neighboring pixels which have a lower distance of 1 from the center pixel C, get labeled and the labeled pixels are inserted in the FIFO queue for recursive label propagation in the non-minima plateau. The label propagation from the present gray level h stops when the queue Q_I is empty. Next, the labeled pixels with gray value h+1 are put into the queue Q_I. When all the gray levels have been visited (with $h = h_{max}$), the procedure of label propagation of the catchment basins from its regional minima is considered to be complete.

To decide the labels of a pixel N_i, $i = 1, \ldots, 8$, the 8-connected neighbors of N_i (which are also

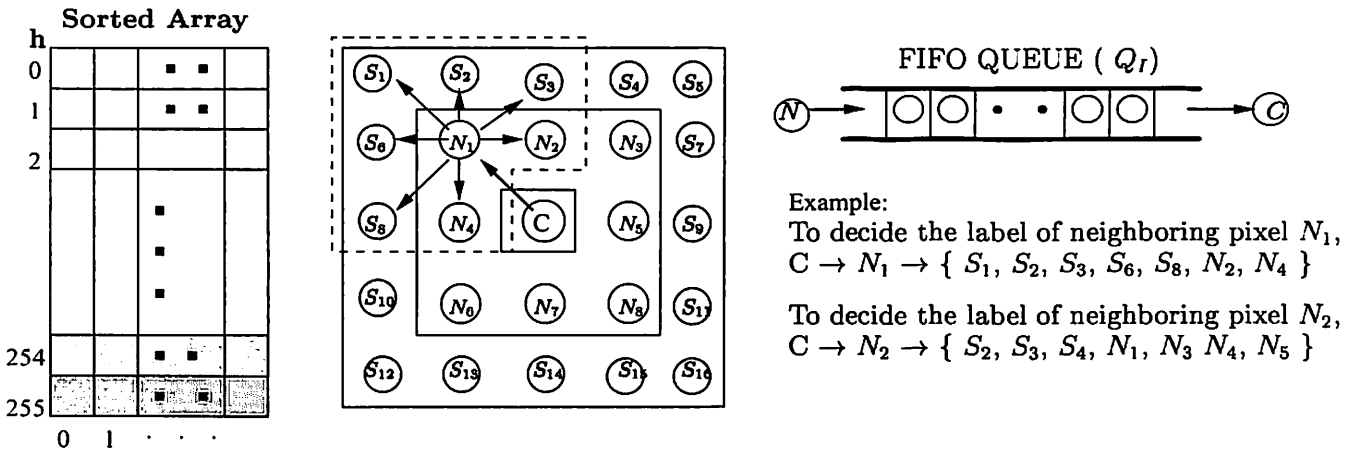


Figure 5.7: Label propagation in proposed algorithm

the neighbors of the center pixel C) and the 8-connected supporting pixels of N_i are considered and processed as follows. The label of the pixel N_1 say is decided as follows. If the condition (viz. the gray level of N_1 with NARM or WSHED (watershed) label being greater than or equal to that of the center pixel C) is satisfied, then one would consider the neighboring pixels namely N_2, N_4, S_1, S_2, S_3, S_6 and S_8 which would then be compared with the pixel N_1. As illustrated in Figure 5.8, four different cases can occur for deciding the label of a pixel p, which are as follows:-

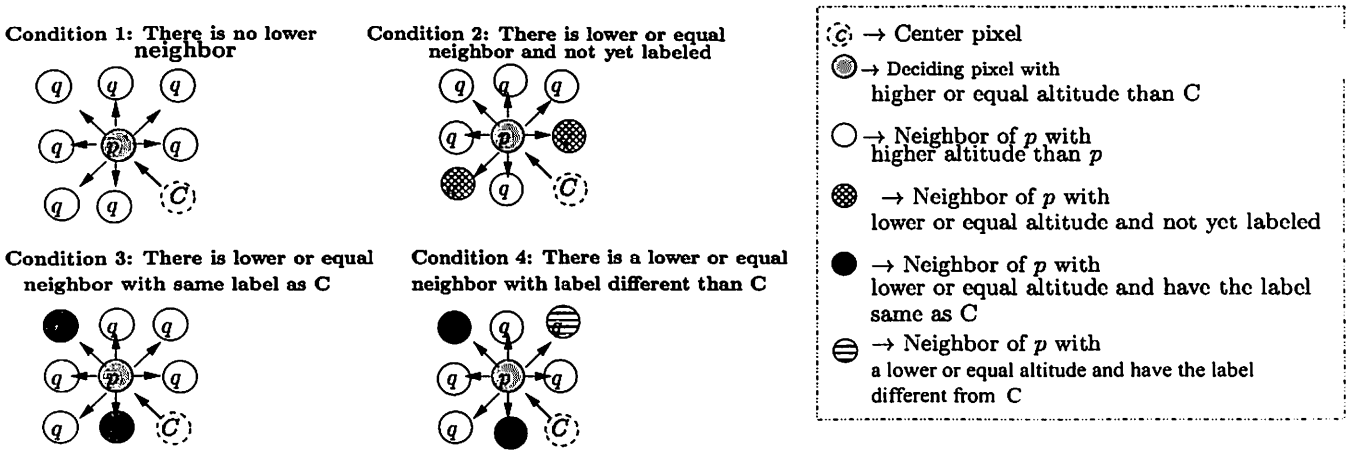


Figure 5.8: Illustrating the four conditions, which can occur during the label propagation from center pixel C to its neighboring pixel p

1. If p has no adjacent pixel with lower altitude, then the pixel p will acquire the label from C.

2. If p has adjacent neighboring pixel(s) q with equal or lower altitude, which is (are) not yet labeled, then the pixel p is labeled with WSHED label, indicating that it belongs to a watershed line.

3. If p has adjacent neighboring pixel(s) q with equal or lower altitude and q has (have) the same label as that of the center pixel C, then the pixel p will obtain the label from pixel C.

4. If p has at least one adjacent neighboring pixel q with equal or lower altitude, which has a label different from the center pixel C, then the WSHED label is assigned to p.

The pseudo-code of the immersion phase of the proposed algorithm is presented next.

Procedure 5.3 *Watershed_by_Modified_Immersion(F, L, d)* */* procedure for watershed transformation by modified immersion following definition 5.5 */*

Input: Morphological multi-scale gradient image F, distance image d and label image L. $\{F, d, L : D_F \rightarrow \mathbb{N}\}$

Output: label image L $\{L : D \rightarrow \mathbb{N}\}$

Let G be a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$ and $N_G(p)$ stands for the neighbors of a pixel p on the grid G

NARM $\leftarrow -1$; /* Not a regional minima */

WSHED $\leftarrow 0$; /* watershed pixel value */

for all h such that $h_{min} \leq h \leq h_{max}$ {/* geodesic SKIZ of level $h-1$ inside level h */} **do**

  for all $p \in D_F$ with $(F(p) = h)$ and $L(p) > 0$ {/* for all labeled pixels at height h */ } **do**

    FIFO_INIT(Q_I) /* initialize the queue Q_I */

    Flag $\leftarrow$ FALSE;

    for all $q \in N_G(p)$ **do**

      if ($L(q)$ = WSHED $\vee$ $L(q)$ = NARM) **then**

        Flag $\leftarrow$ TRUE;

      end if

    end for

    if Flag = TRUE **then**

      FIFO_ADD(p, Q_I); /* enqueue the pixel p having at least one NARM or WSHED neighboring pixel into queue Q_I */

    end if

  end for

  while FIFO queue Q_I not empty {label propagation at level h} **do**

21: $p \leftarrow$ FIFO_REMOVE(Q_I); /* Dequeue the pixel p from queue Q_I; */
22: **for all** $q \in N_G(p)$ **do**
23: **if** $F(q) > F(p)$ and its distance $d(q)$ is equal to 1 **then**
24: Decide_Label_q(F, L, p, q); /* procedure for deciding the label of the pixel q based on four different cases (as illustrated in Fig. 5.8) */
25: **else if** $F(q) = F(p)$ and its lower distance $d(q) = d(p)+1$ **then**
26: Decide_Label_q(F, L, p, q); /* procedure for deciding the label of the pixel q based on four different cases (as illustrated in figure 5.8) */
27: FIFO_ADD(q, Q_I); /* enqueue the pixel q into queue Q_I for label propagation in non-minima plateau region */
28: **end if**
29: **end for**
30: **end while**
31: **end for**
32: /* end of procedure Watershed_by_Modified_Immersion */

Next, the procedure Decide_Label_q used in procedure Watershed_by_Modified_Immersion may be given in pseudo-code form as follows.

1: **Procedure 5.3.1** *Decide_Label_q(F, L, p, q); /* procedure for deciding the label of the pixel q based on four different cases (as illustrated in Fig. 5.8) */*
2: **Input:** F, L, labeled pixel p and deciding pixel q.
3: **Output:** Decided pixel label $L(q)$.
4: $Flag_1 \leftarrow$ FALSE; $Flag_2 \leftarrow$ FALSE; /* conditional comparison flags */
5: **for all** $r \in N_G(q)$ and $(r \neq q)$ {/* Inspect all the supporting neighbors of q except p */} **do**
6: **if** $F(r) < F(q)$ {/* pixel r have gray value less than q */} **then**
7: **if** $L(r) > 0$ {/* pixel r is already labeled */} **then**
8: $Flag_1 \leftarrow$ TRUE;
9: **if** $L(r) \neq L(p)$ {/* pixel r have label different from p} **then**
10: $Flag_2 \leftarrow$ TRUE;
11: **else if** $L(r) < 0$ and $F(r) \geq F(p)$ {/* pixel r not yet labeled and its gray value $F(r)$ is greater than or equal to $F(p)$ */} **then**
12: $Flag_2 \leftarrow$ TRUE;

```
13:         end if
14:       end if
15:     else if F(r) = F(q) {/* pixel r has gray value equal to that of q */} then
16:       if L(r) > 0 {/* pixel r is already labeled */} then
17:         Flag_1 ← TRUE;
18:         if L(r) ≠ L(p) {/* pixel r have label different from p */} then
19:           Flag_2 ← TRUE;
20:         end if
21:       end if
22:     end if
23: end for
24: if Flag_1 = FALSE {/* pixel q has no adjacent pixel with lower altitude*/ } then
25:   L(q) ← L(p);   /* pixel q acquires the label from p */
26: else if Flag_2 = FALSE {/* pixel q has adjacent pixel r with equal or lower altitude and it has the
    same label as that of the pixel p */} then
27:   L(q) ← L(p);   /* pixel q acquires the label from p */
28: else if Flag_2 = TRUE {pixel q has at least one adjacent neighboring pixel r and has the label
    different from the pixel p */} then
29:   L(q) ← WSHED;   /* pixel q is labeled as a watershed pixel */
30: end if
31: /* end of procedure Decide_Label_q */
```

Performance of Flooding Procedure– Let the performance of the label propagation of the simulated immersion procedure be defined as the ratio of the number of pixels whose labels are decided at one point of time to the total number of pixels which are necessary to achieve this. For Vincent's algorithm [VS91], the ratio is seen as 1/9. Whereas, as depicted in Fig. 5.7, for the proposed algorithm, this ratio turns out to be 8/25. For, based on all the twenty-five pixels shown in the figure, the labels of the eight pixels N_1, N_2, $\cdots$, N_8 are determined. Hence, there is an improvement of performance from $\frac{1}{9}$=11% to $\frac{8}{25}$ = 32%.

5.3 Complexity Analysis

Let us denote by n the total number of pixels in the gray scale image F whose domain is denoted as $D_F \subset \mathbb{Z}^2$, and G is a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$. $N_G(p)$ stands for the neighbors of a pixel p on the grid G. Let $p_1, p_2, \ldots, p_P$ be P plateaus, each p_i having n_i pixels and P_N be the number of detected non-minima plateaus.

Local Minima Detection and Labeling

Total number of comparisons for Local minima detection and labeling (LMD) (on an average) is

$$LMD = N_G * \left(\sum_{i=1}^{P} n_i + \sum_{j=1}^{P_N} n_j \right) \approx n$$

where N_G stands for neighborhood pixels on the grid G.

Sorting step

The present sorting technique has the advantage of requiring $2n$ "look and do" operations [VS91] – one for determining the frequency distribution and the other for the assignment of pixel addresses.

Flooding process

Let $l_0, l_1, \ldots, l_{H-1}$ be the H connected gray levels in the image F; moreover, let l_j have m_j pixels which have at least one NARM neighboring pixel. Let C be the present center pixel and $N_1, N_2, \ldots, N_8$ be the 8-connected neighbors of the center pixel. The number of comparisons required for deciding the labels of the neighboring pixels ($N_1 - N_8$) of the center pixel C by using conditional comparisons is

$$N_G(C) + \sum_{i=1}^{8} N_G(N_i) - \{C\}$$

Thus the total number of comparisons for the entire immersion process (IP) (in the worst case) is

$$IP = \left(N_G(C) + \sum_{i=1}^{8} N_G(N_i) - \{C\} \right) * \sum_{j=0}^{H-1} m_j \approx n$$

The computational complexity of the proposed watershed algorithm is then

$$\begin{aligned} &= 2n + LMD + IP \\ &= 2n + 2n \text{ (on an average)} = 4n \approx O(n) \end{aligned}$$

In the above analysis, the factor $4n$ is accounted for by $2n$ operations in sorting step followed by $2n$ operations for the minima detection step and the immersion step. Since the above three steps run in linear time, the entire algorithm is linear with respect to n.

Analysis of conventional algorithm [VS91]

The number of comparisons in the sorting step are the same as that in the proposed algorithm. During the flooding process, however, each pixel is scanned thrice on an average in case of Vincent-Soille algorithm [VS91]. Thus the computational complexity of the conventional watershed algorithm (on an average) is

$$= 2n + 3n = 5n \approx O(n).$$

From the above analysis, the proposed algorithm requires $4n$ comparisons as against $5n$ comparisons for the conventional algorithm.

5.4 Experimental results

The improved algorithm has been simulated under SUN Solaris 8 operating system with 750 MHz Ultra SUN SPARK III Computer. Watershed transformation usually produces over-segmentation due to additive noise in natural scene images. To alleviate this problem, each image data was first pre-processed and the multi-scale gradient operator [Wan97] with different threshold values (h) was applied, which effectively enhanced the blurred edges and reduced the number of local minima. Table 5.1 shows that the proposed improved algorithm is more efficient than the existing algorithm due to Vincent [VS91]. It may be mentioned here that the presence of a large number of plateaus in case of some images like Table Tennis and Road traffic accounts for considerably less computation time by the proposed algorithm relative to Vincent-Soille algorithm, as shown in Table 5.1. Also, the segmentation results are obtained for various threshold values (h). Figure 5.9 shows that an increase in the threshold value (h) leads to a reduction in the number of homogeneous regions. Figure 5.10 depicts the variation of the computation time while running the proposed and the conventional algorithms for different values of h. The results of Peppers, Washington_ir, Lena, Table Tennis, Road Traffic and Blood cells images and their corresponding watershed images are shown in Figures 5.11, 5.12, 5.13, 5.14, 5.15 and 5.16 respectively.

Following a region-based approach [HD95] (as mentioned in Sec. 3.3.1 Chapter 3) to quantitatively evaluate the performance of a segmentation algorithm, the performance measure P of the proposed

Test image	Threshold value (h)	Number of Regions (R)	Time (Sec)	
			Vin'91	Proposed
Lena(256×256)	10	64	0.37	0.35
Table Tennis(352×240)	12	25	0.42	0.24
Peppers(256 × 256)	15	58	0.38	0.36
Cermet(256 × 256)	30	61	0.37	0.25
Washington_ir(250 × 250)	16	46	0.31	0.25
Head(176 × 172)	10	36	0.14	0.11
Road Traffic(256 × 256)	20	48	0.46	0.30
Blood Cells(350 × 212)	26	20	0.20	0.19

Table 5.1: Computation times (in sec.) for different image data on SUN Solaris 8 operating system with 750 MHz Ultra SPARC III.

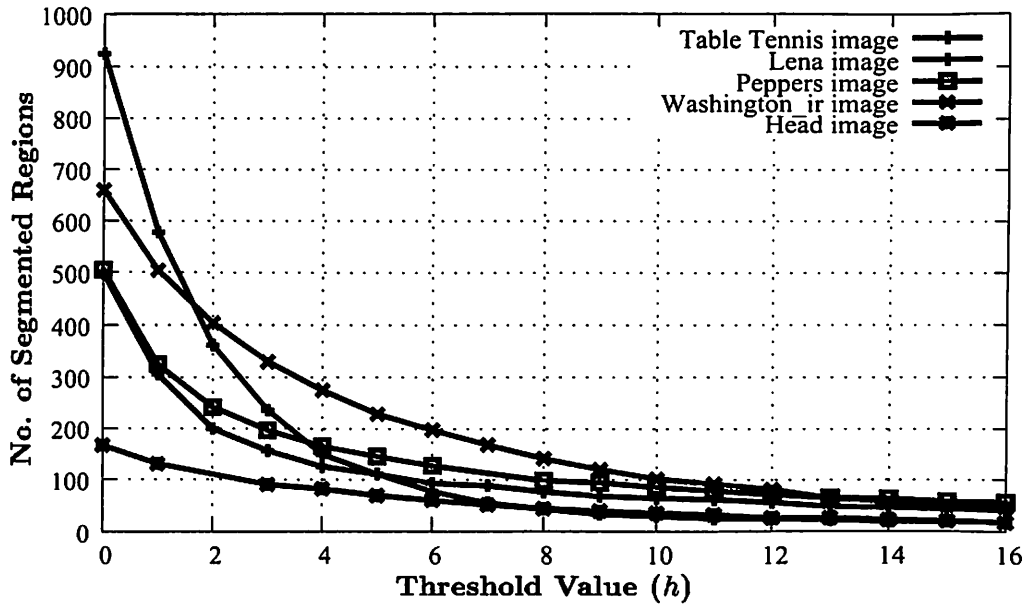


Figure 5.9: Number of homogeneous regions for different threshold value (h).

algorithm evaluated for various test images is provided in Table 5.2, which shows that the homogeneous regions are identified by both algorithms in an almost identical manner ($P \approx 1$). While calculating $D_H(S_1 \Rightarrow S_2)$, the number of discontinuous pixels obtained out of the proposed algorithm with respect to Vincent's algorithm [i.e. column 3 of Table 5.2] has to be added and while calculating the reverse distance $D_H(S_2 \Rightarrow S_1)$, the number of discontinuous pixels obtained out of Vincent's algorithm with respect to proposed algorithm [i.e. column 4 of Table 5.2] has to added. From the segmentation results,

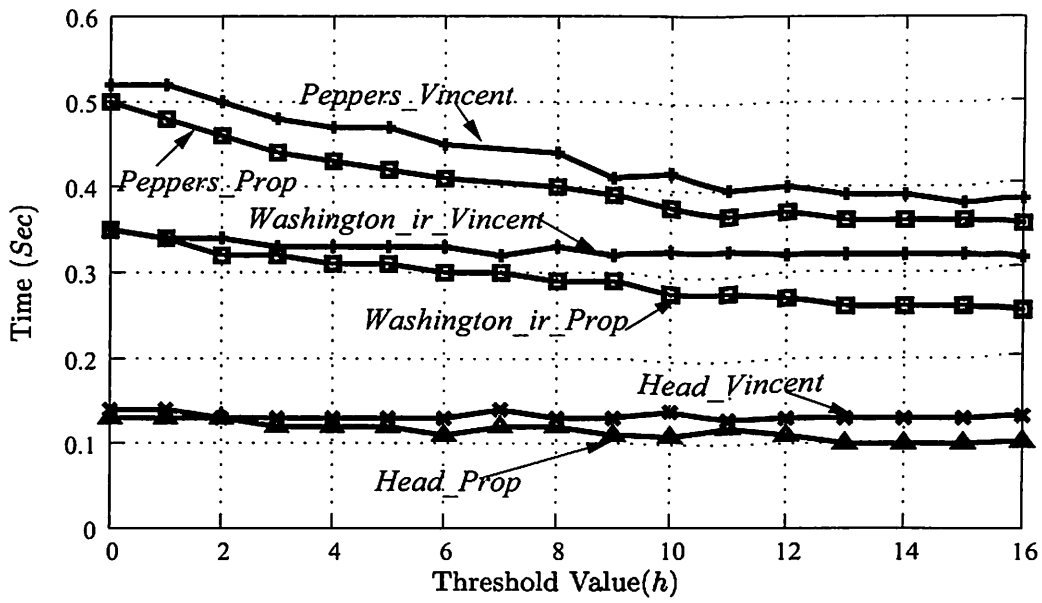
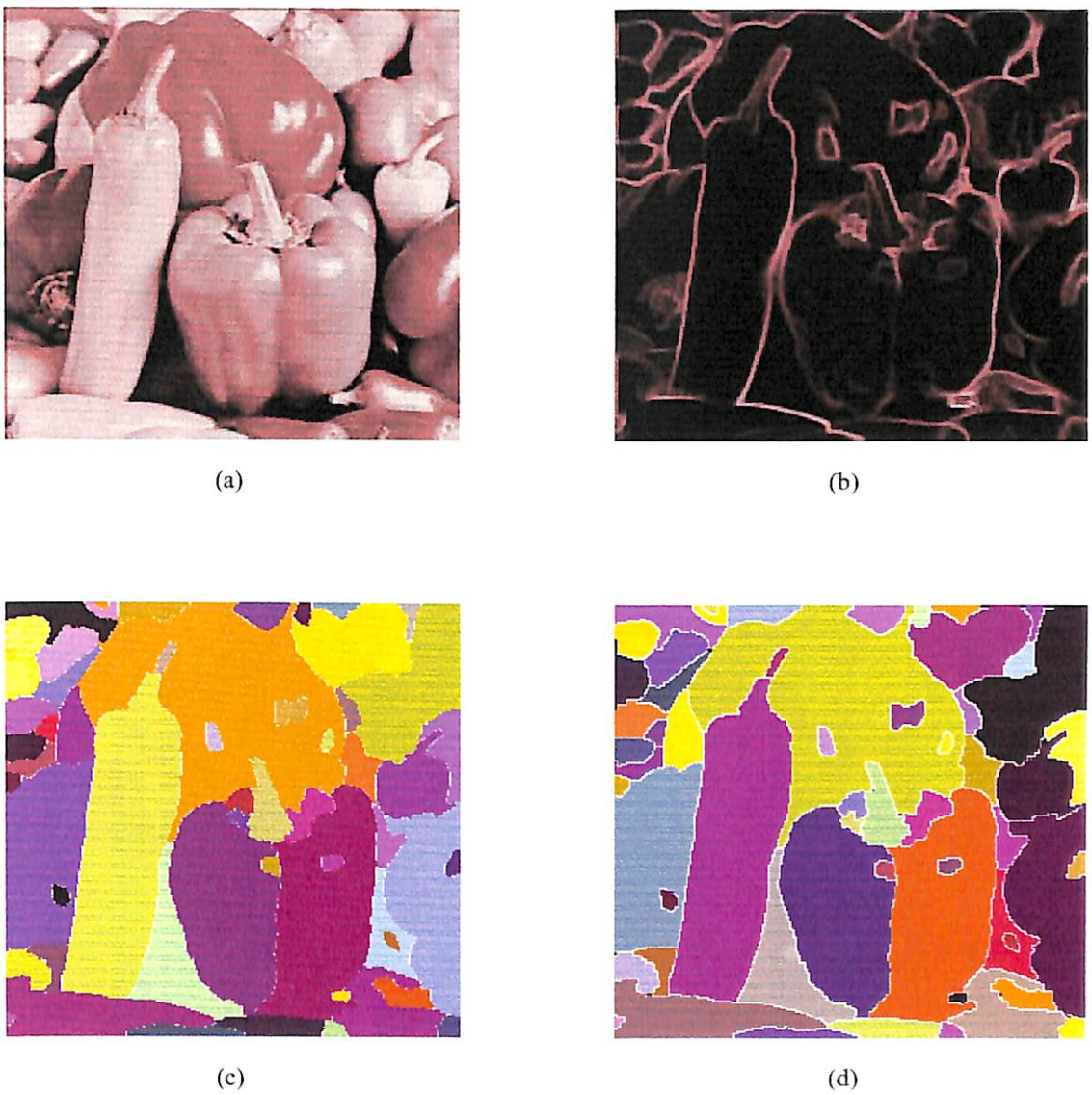


Figure 5.10: Computation time while running the proposed and conventional algorithm due to Vincent for different threshold value (h).

the continuous and thin closed contours can be clearly identified.

Test Image	(h)	Number of regions	No. of discontinuous pixels resulting from		Performance measure (P)
			proposed algo. *w. r. t.* Vincent's algo.	Vincent's algo. *w. r. t.* proposed algo.	
Lena (256×256)	10	64	207	2943	0.975998
Tennis (352×288)	12	25	105	1643	0.989654
Peppers (256×256)	15	58	177	2869	0.976761
Cermet (256×256)	30	61	62	3389	0.973671
Washinton_IR (250×250)	16	46	171	2220	0.980872
Head (176×172)	10	36	97	1321	0.976579
Road Traffic (256×256)	20	48	121	1034	0.991180
Blood Cells (212×353)	26	20	84	2253	0.984252

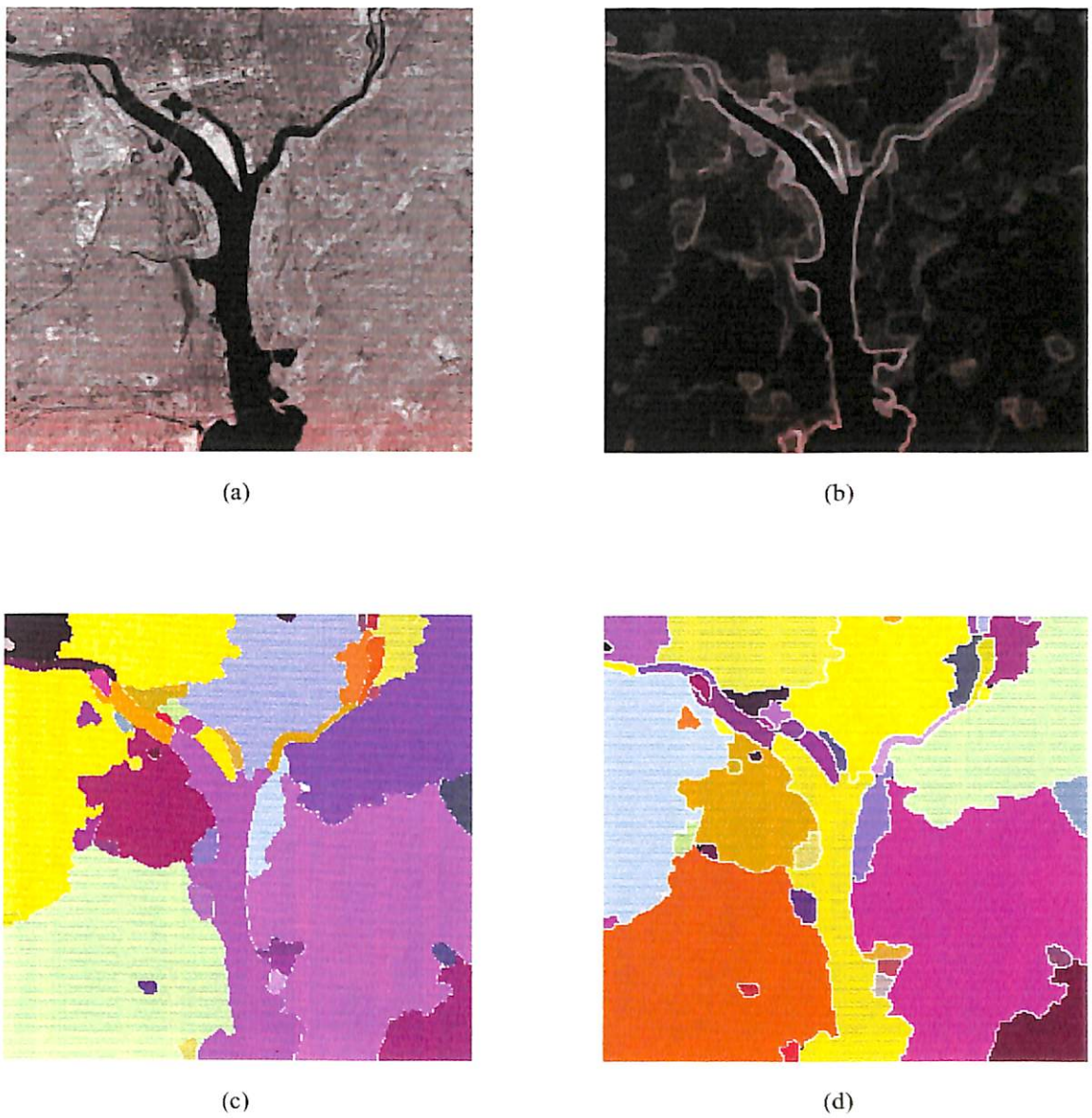
Table 5.2: Quantitative evaluation of the segmented images



(a) (b)

(c) (d)

Figure 5.11: (a). Original Peppers image. (b). Morphological multi-scale gradient operation on Peppers image with a threshold value of 12. (c). Segmentation produced by Vincent's watershed algorithm [VS91]. (d). Segmentation produced by the proposed watershed algorithm



(a) (b)

(c) (d)

Figure 5.12: (a). Original Washington_ir image. (b). Morphological multi-scale gradient operation on Washington_ir image with a threshold value of 16. (c). Segmentation produced by Vincent's watershed algorithm [VS91]. (d). Segmentation produced by the proposed watershed algorithm

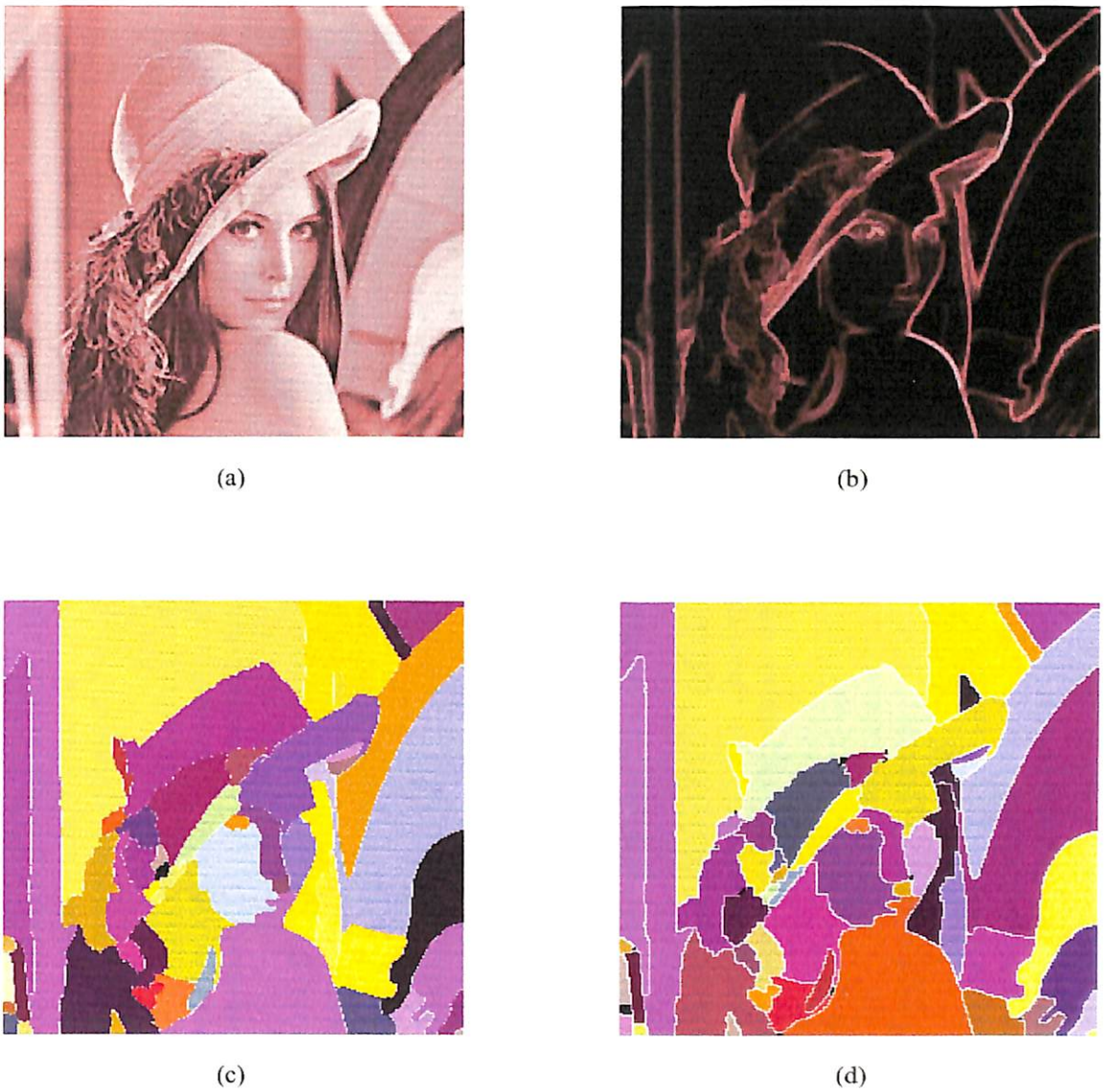
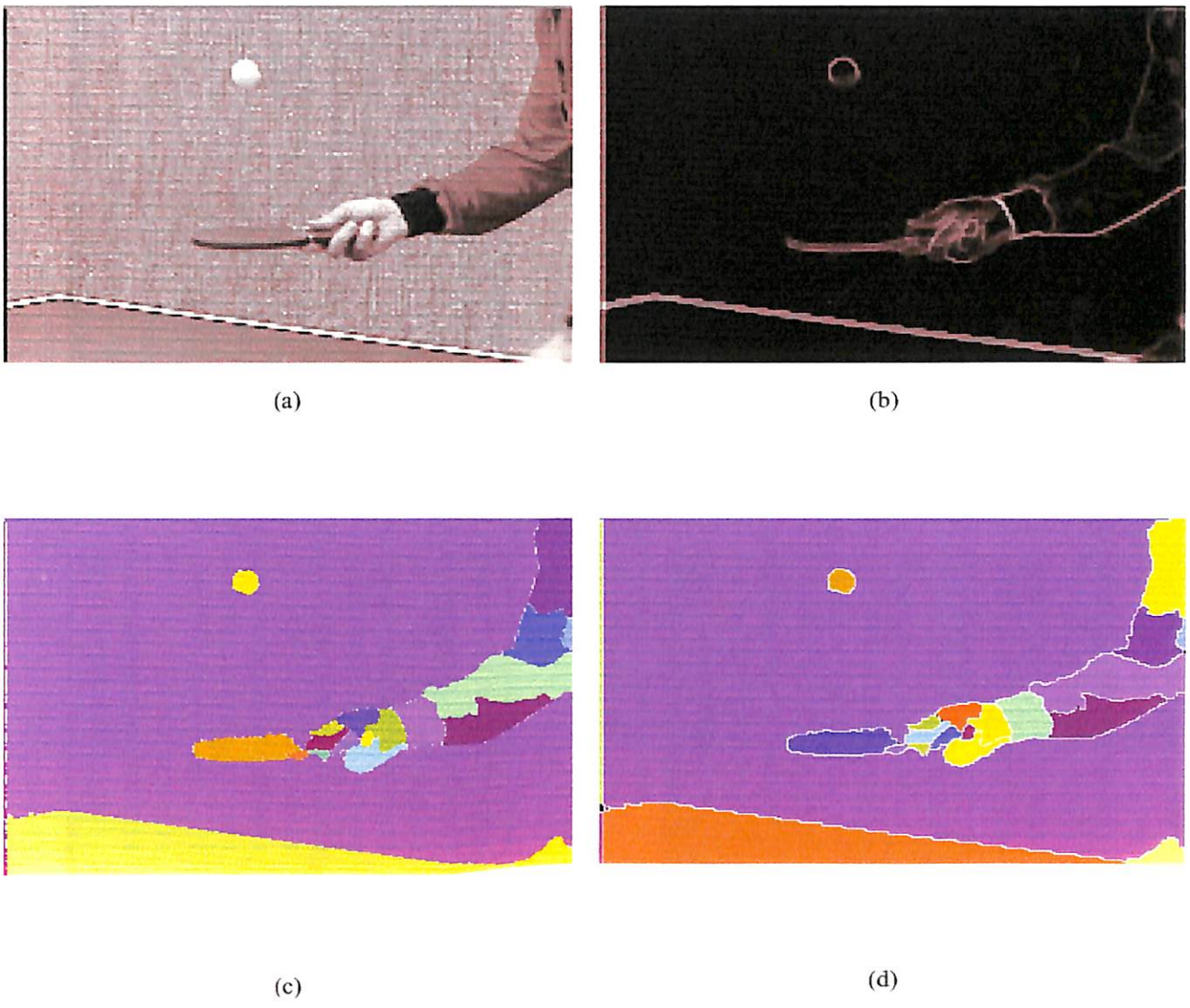


Figure 5.13: (a). Original Lena image. (b). Morphological multi-scale gradient operation on Lena image with a threshold value of 12. (c). Segmentation produced by Vincent's watershed algorithm [VS91]. (d). Segmentation produced by the proposed watershed algorithm



(a) (b)

(c) (d)

Figure 5.14: (a). Original Table Tennis image. (b). Morphological multi-scale gradient operation on Tennis image with a threshold value of 12. (c). Segmentation produced by Vincent's watershed algorithm [VS91]. (d). Segmentation produced by the proposed watershed algorithm

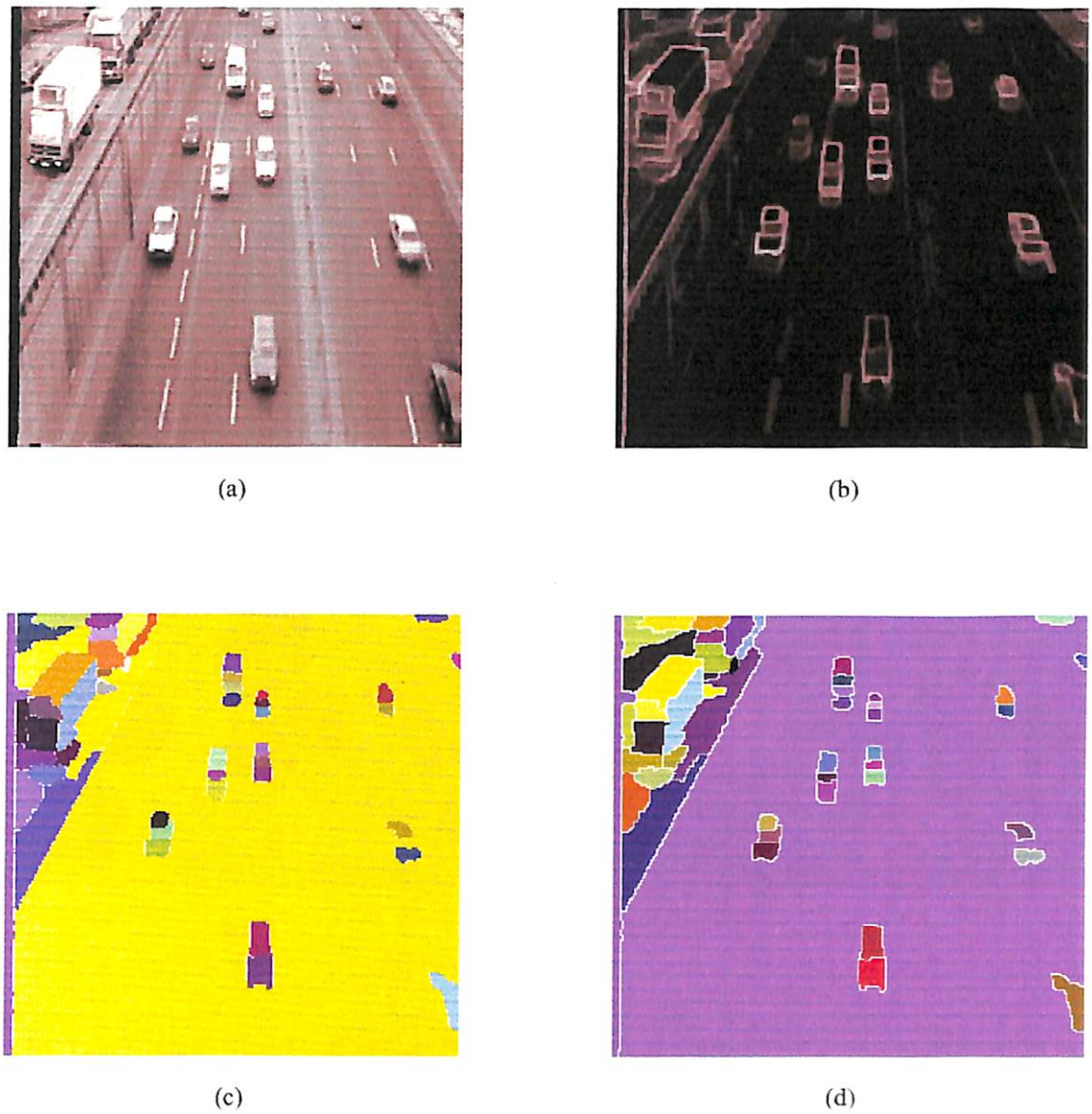


Figure 5.15: (a). Original Road traffic image. (b). Morphological multi-scale gradient operation on Road traffic image with a threshold value of 20. (c). Segmentation produced by Vincent's watershed algorithm [VS91]. (d). Segmentation produced by the proposed watershed algorithm

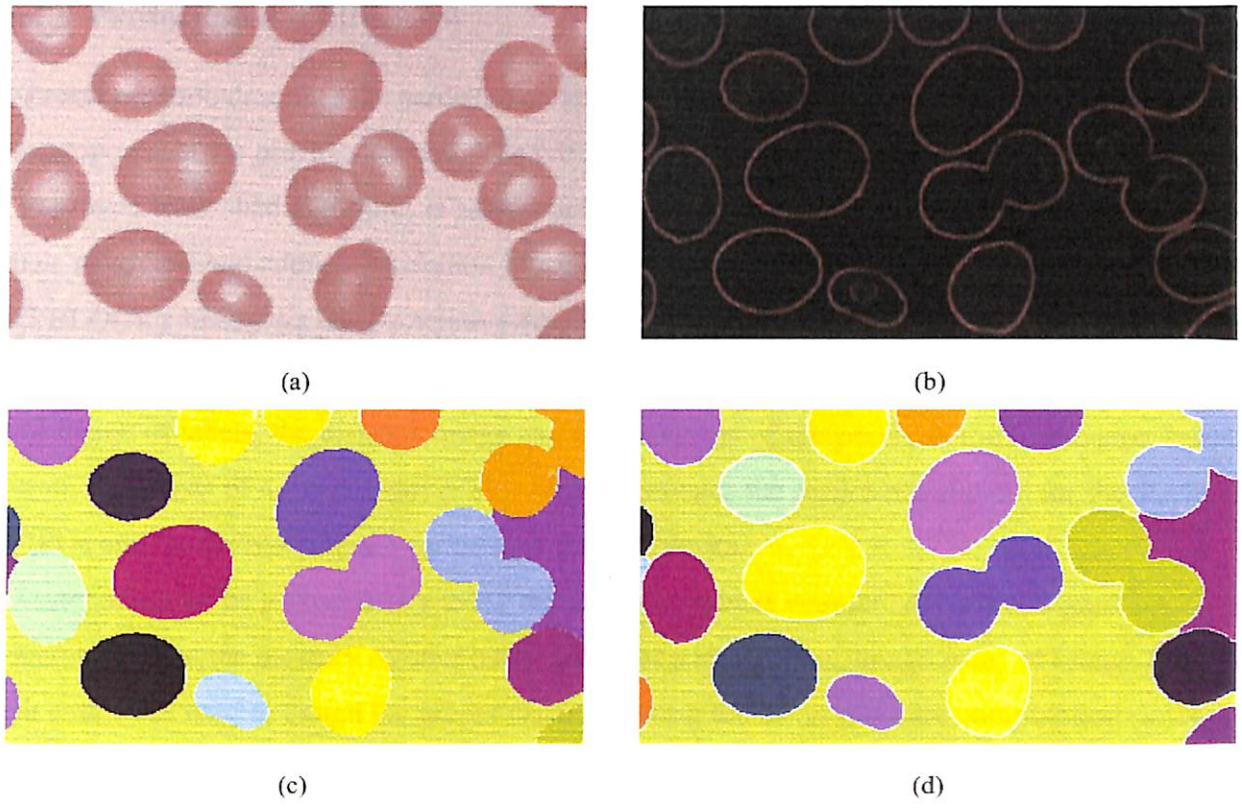


Figure 5.16: (a) Original Blood cell image; (b) Morphological multi-scale gradient operation on Blood cell image with a threshold value of 26; (c) Segmentation produced by Vincent's watershed algorithm [VS91]; (d) Segmentation produced by the proposed watershed algorithm.

5.5 2-D Cellular Automata based Prototype Architecture

This section presents a detailed discussion on a prototype architecture for implementing the improved immersion-based watershed algorithm proposed in this chapter. Moreover, an FPGA implementation of the architecture is briefly described.

5.5.1 Prototype Architecture

The present section describes the hardware architecture that implements the improved watershed algorithm proposed in the previous section. The block diagram of the overall architecture that implements the improved watershed algorithm is shown in Figure 5.17. This architecture consists of two FIFO queues, neighborhood address generation block (NAG BLOCK), supporting address generation block (SAG BLOCK), histogram and cumulative distribution block, plateau detection block, plateau analysis block, lower distance block and four memory modules, which are meant to hold the input image, the output image of labels, the distance image and the addresses of the pixels as the immersion starts. In addition, it consists of a control unit (FSM) which generates the read/write signals for the RAMs and the FIFOs and the set/reset signals for data processing blocks.

Raster scan can be performed by using the memory address counter. A control signal *scan_comp* is generated, when the memory counter reaches the last address (max_image_size) of image. The wave length counter is used to assign the proper distance during the lower distance computation process. In addition, the gray level counter is used to select the pixel frequency during the process of computation of the cumulative distribution. The NAG BLOCK generates the eight neighborhood addresses of the center pixel. The SAG BLOCK generates the addresses of the sixteen supporting pixels from the addresses of eight neighboring pixels which are generated by the NAG BLOCK. The plateau detection block detects the neighbors that have the same intensity value as that of the center pixel. The detected pixels which can be further explored are enqueued in FIFO_Q1. The plateau analysis block compares the center pixel with the neighboring pixels and decides the label of the plateau. The histogram and cumulative distribution block is used to sort the pixel addresses according to the ascending order of gray levels. The conditional neighbor comparison block decides the labels of the neighboring pixels based on four different cases discussed in Section 5.2.2. Parallelism has been explored in this block which accommodates the eight processing elements (PE) involved in comparing both the intensities of the neighboring pixels and the supporting pixels with that of the corresponding center pixel.

The sequence of operations taking place in the data processing unit is controlled by the control

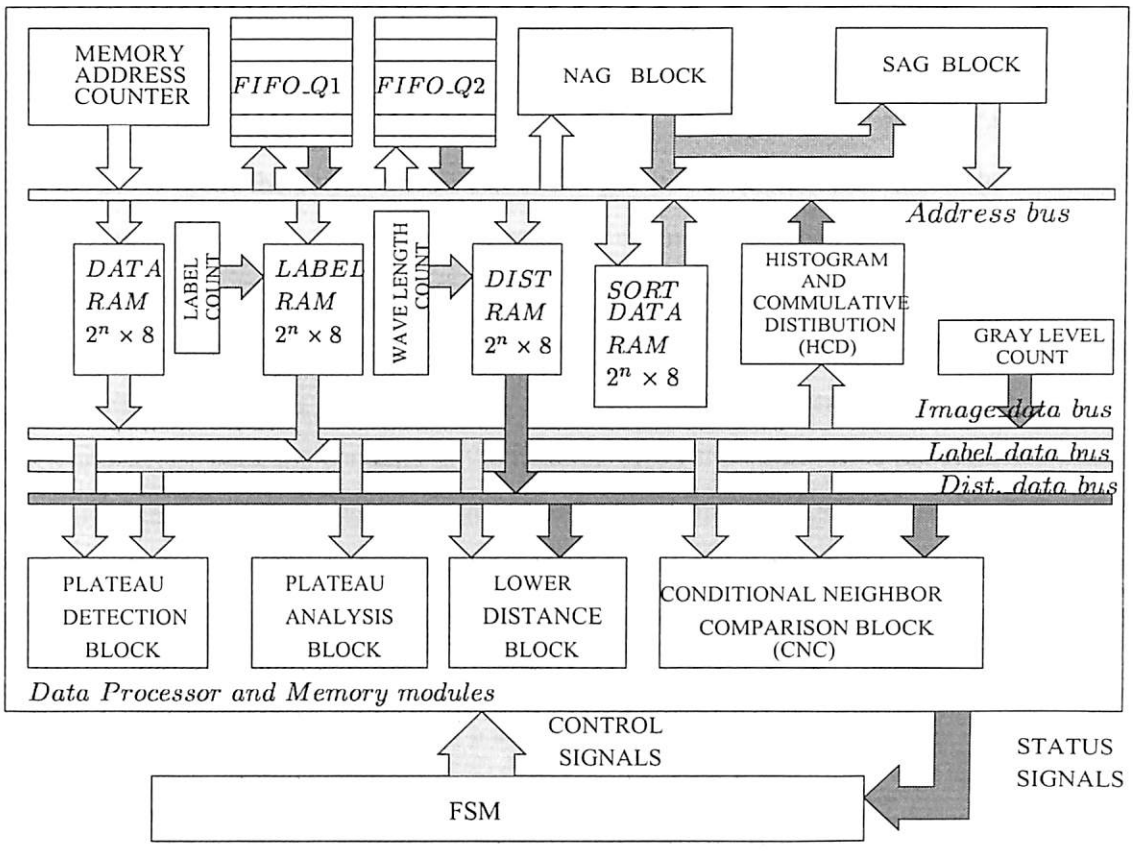


Figure 5.17: Functional block diagram of the proposed watershed transform architecture

signals generated by the control unit which is a finite state machine (FSM). The control signals fall into three broad categories, which are concerned with the tasks of memory address generation, data transfer and processing. The data on the data bus is transferred under the control of the data transfer signals to different registers present in the data processing blocks. The control signals for processing includes those for achieving the jobs of plateau detection, plateau analysis, histogram and cumulative distribution, lower distance and conditional neighbor comparison. The principal functional modules of the architecture are described as follows.

Histogram and Cumulative Distribution block

The various labeled blocks following the DATA_RAM namely H_COUNT, DECODER 1×256, Register bank(RB), INR, ACC and FSM are grouped as the histogram and cumulative distribution block (dotted line in Figure 5.18). The histogram and cumulative distribution process consists of two sequential stages. The process begins by accumulating the occurrences of the pixel gray values in the

Register bank (RB). Next, the computation of cumulative distribution is performed. The blocks labeled ADDR_COUNT, DATA_RAM, DECODER, RB, MUX, and INR are used for histogram computation. The register bank (RB) is addressed by the gray scale value of the pixel from the DATA_RAM. The increment (INR) block reads the data from the selected register, increments and writes back the contents of the register. The read-modify-write cycle operation consumes two clock cycles.

The labeled blocks H_COUNT, RB, DECODER, MUX, and ACC are used for accomplishing the cumulative distribution process. The H_COUNT is used to address the register bank (RB) during the cumulative distribution process. The accumulator (ACC) reads the data from the selected register, accumulates and writes back the contents of the register. After the cumulative process, the contents of the register bank (RB) provide the starting memory addresses of the pixel gray levels in the SORT DATA RAM. The control unit FSM is a synchronous finite state machine which controls the histogram and cumulative distribution block by providing the appropriate control signals, timing signals and set/reset signals.

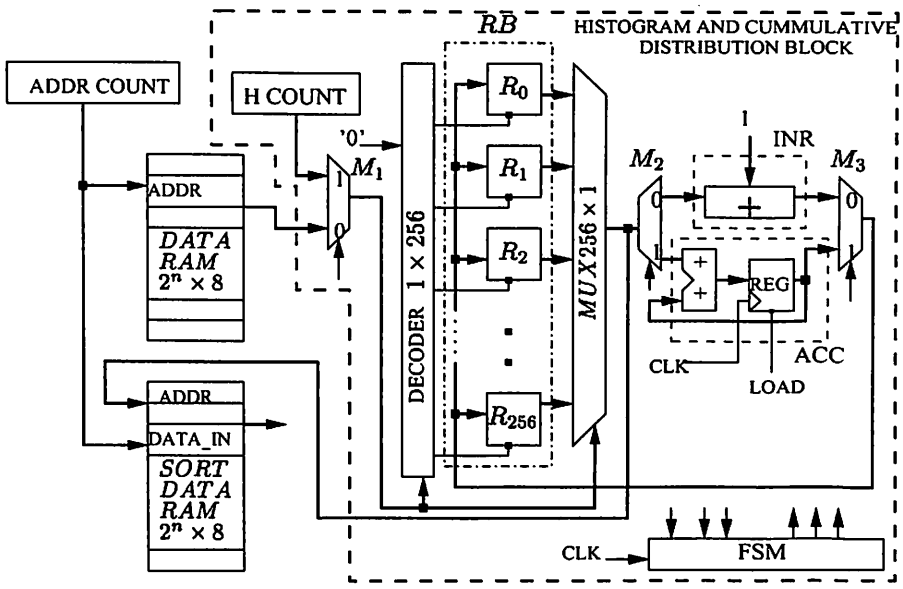


Figure 5.18: Schematic diagram for Histogram and Cumulation Distribution Block.

Total number of clock cycles for histogram and cumulative distribution process is 2*n + 512 where n is the number of pixels in the input image. The performance of Histogram and Cumulative Distribution block is limited to 48.43 MHz in XCV400HQ240. It utilizes 6% of CLB slices and 9% of the registers.

Plateau Detection Block

The plateau detection is performed by comparing the center pixel data (C_DATA) with each of its 8 nearest neighbors (N_DATA_1, $\cdots$, N_DATA_8) in parallel, as depicted in Figure 5.19(a). The schematic diagram of the PD_N (plateau detection for single neighbor) block is shown in Figure 5.19(b). The f_w_flag signal is generated when the neighbor pixel data (N_DATA) and the label image data (N_LABEL) are equal to the center pixel data (C_DATA) and the initialized value of the label image INIT respectively. It indicates that the neighboring pixel belongs to its plateau and not visited previously. This status signal causes the control unit to generate the necessary signal to enqueue the address of the neighbor pixel into FIFO_Q1. The performance of the Plateau Detection Block is limited to 81.05MHz in XCV400HQ240. It utilizes 1% of CLB slices and 1% of the registers.

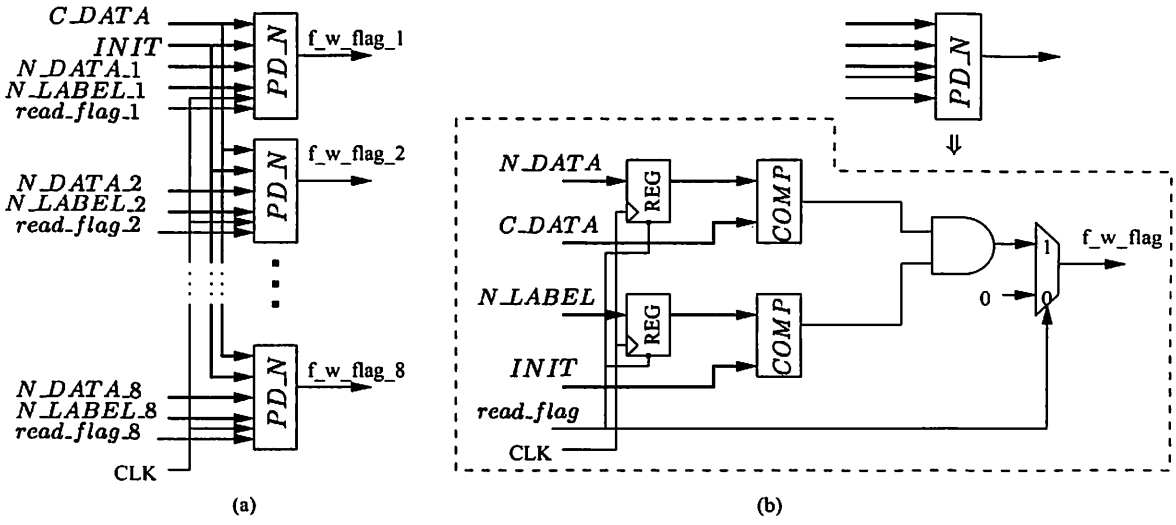


Figure 5.19: (a) Block diagram of the Plateau Detection Block; (b) schematic diagram for PD_N block.

Plateau Analysis Block

The schematic diagram shown in Figure 5.20 is used to detect whether the center pixel belongs to the non-minima plateau. The control signal plateau_analysis_flag is generated when the neighbor pixel data (N_data) is less than the center pixel data (C_DATA) and the center pixel data of distance image (C_DIST) is equal to DIST_INIT. It indicates that the center pixel belongs to the non-minima plateau and it is an *outer* pixel. This control signal invokes to generate the necessary signal to enqueue the address of the center pixel into FIFO_Q2. The performance of the Plateau Detection Block is limited

to 58.36 MHz in XCV400HQ240. It utilizes 1% of CLB slices and 1% of the registers.

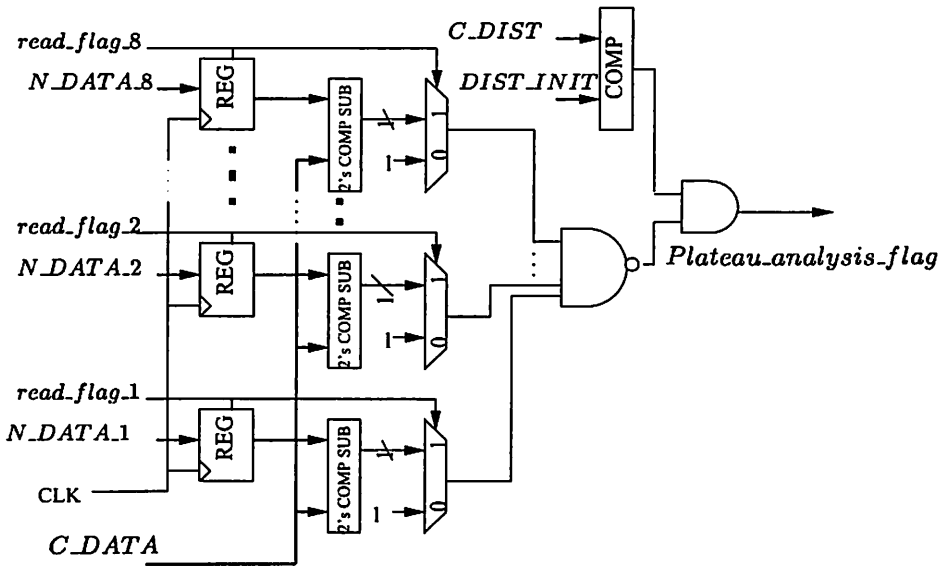


Figure 5.20: The Schematic diagram of the Plateau Analysis Block.

Lower Distance Block

The logic module shown in Figure 5.21 detects whether the neighboring pixel belongs to a non-minima plateau for which no distance is assigned. The control signal l_w_flag is generated when the neighboring pixel data (N_DATA) and the distance image data (N_DIST) are equal to the center pixel data (C_DATA) and the initialized value of the distance image DIST_INIT respectively. It indicates that the neighboring pixel belongs to a non-minima plateau and no distance is assigned to it. This status signal invokes the control unit to generate the necessary signal to enqueue the address of the neighboring pixel into FIFO_Q2 for further exploration and assignment of the current distance value of the WAVE_LENGTH_COUNT to the distance image pixel. The performance of the Lower Distance Block is limited to 62.33MHz in XCV400HQ240.

Conditional Neighborhood Comparison Block

The conditional neighborhood comparison block decides the labels of the neighboring pixels based on the neighborhood comparison. The (3 × 3) 2-D CA (2-dimensional cellular automata) is used in the conditional neighborhood comparison process as shown in Figure 5.22(a). It accommodates the cells

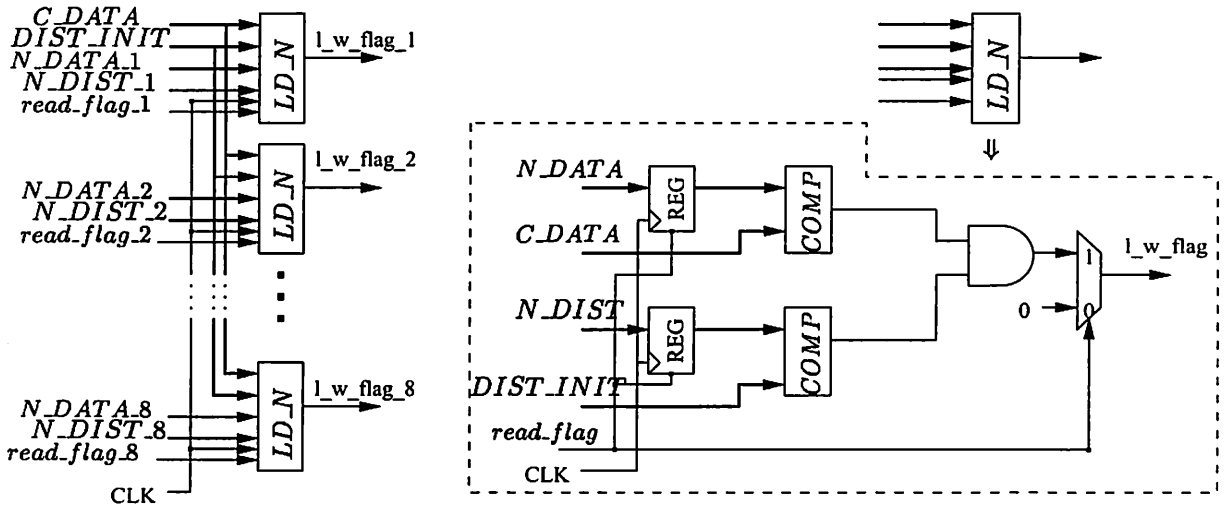


Figure 5.21: The Schematic diagram of the Lower Distance Block.

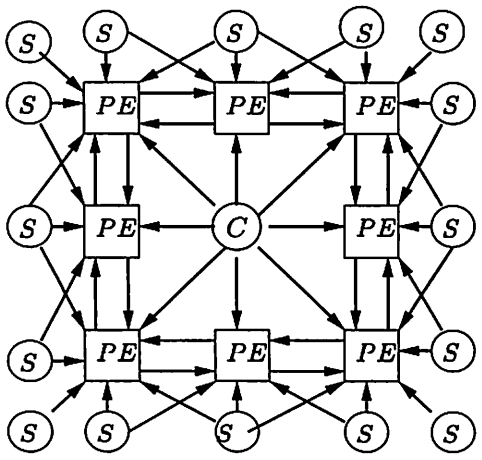
(PE) involved in comparing the intensities as well as the labels of the neighboring pixels (from the adjacent cells) and the supporting pixels (S) with that of the center pixel (C).

Cellular automata [CCNC97] are a special class of automata that are made of simple cells each of which is a simple finite state machine. A 2-D CA is a generalization of a 1-D CA, where the cells are arranged in a two-dimensional grid with connections among the neighboring cells. The states of the cells are updated in parallel according to a local rule. That is, the state of a cell at a given time step depends only on itself and the states of its eight nearest neighbors existing at the previous time step. The neighborhood function specifies the next state of the cell $C_{i,j}$ in terms of itself and the states of its 8 nearest neighbors (8-neighborhood dependency, Figure 5.22(b)). That is, the state of the $(i,j)^{th}$ cell of a 2-D CA is given by

$$C_{i,j}^{t+1} = f(C_{i,j}^{t}, C_{i-1,j-1}^{t}, C_{i-1,j}^{t}, \cdots, C_{i+1,j-1}^{t}, C_{i+1,j}^{t}, C_{i+1,j+1}^{t}) \tag{5.5.1}$$

Since f is a combinational function of 9 variables, all the cells on the 2-D lattice are updated simultaneously.

One may consider here a fully parallel synchronous (3×3) 2-D CA model with eight processing elements for eight neighbors of the center pixel. The functional block diagram of a typical cell (PE) is given in Figure 5.23. The data bus loads the input data to a BUFFER inside a PE. The labeled C_N module block is used to detect the validity of the deciding neighboring pixel (N_DATA). The validity signal c_n_comp_flag is generated when the neighboring pixel (N_DATA) has its intensity value greater than that of the center pixel (C_DATA), the distance (N_DIST) equal to one more than the center



(a)

TOP LEFT (i−1,j−1)	TOP (i−1,j)	TOP RIGHT (i−1,j+1)
LEFT (i,j−1)	SELF (i,j)	RIGHT (i,j+1)
BOTTOM LEFT (i+1,j−1)	BOTTOM (i+1,j)	BOTTOM RIGHT (i+1,j+1)

(b)

Figure 5.22: (a). The 3×3 2-D CA for conditional neighborhood comparison. (b). Eight-Neighborhood Cell Dependency

pixel distance (C_DIST) and the neighboring pixel label (N_LABEL) is equal to NARM (not a regional minima). The blocks N_N_1, N_N_2, ..., N_N_7 are used to decide the label of the neighboring pixel (N_DATA) based on four different conditions (as depicted in Figure 5.8). A load_flag signal is generated when one of the four conditions is satisfied. This enables the state register to load the neighbor pixel label (N_LABEL) with the center pixel label (C_LABEL). The performance of the conditional neighborhood comparison block is limited to 44.46 MHz in XCV400HQ240.

5.5.2 FPGA implementation

The proposed watershed architecture has been modelled in VHDL [vhd93] and then implemented in XILINX Virtex FPGA device. An image size greater than 30 × 30 requires more than one chip for its implementation. Due to limitations of the memory on the virtex device, the performance of the FPGA has been limited to an image of the size 30 × 30. So, in this design the block RAMs are configured with 8-bit data lines and 9-bit address lines. Tests were run assuming a clock frequency of 40 MHz The simulation results obtained for various test images are given in Table 5.3. From the simulation results, the hardware implementation is seen to be faster than the software version by an order of 3. The software results shown in Table 5.3 are achieved using Linux 9.0 environment on a Pentium IV 1.5 GHz processor system.

The entire design is simulated in Modelsim Xilinx 5.5b, synthesized at gate level and then imple-

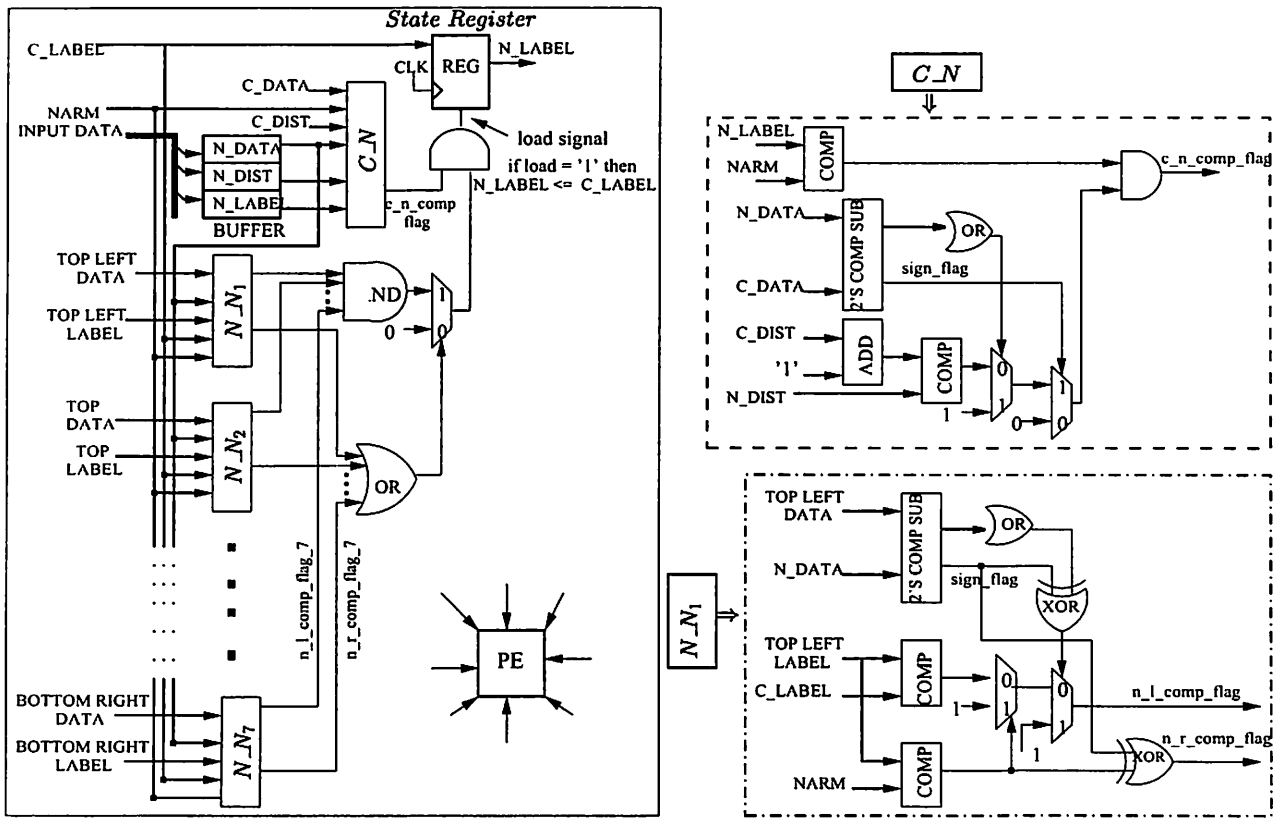


Figure 5.23: The schematic diagram for deciding the label of a neighboring pixel (PE)

Test Image (30×30)	Software Time (*millisecond*)	Hardware Time(μ*second*)
Claire	16	48.15
Tennis	16	39.17
Heart	12	44.19
Akiyo	10	38.67
MRI Brain	20	51.53
Cermet	20	39.12

Table 5.3: Comparison between simulation time of hardware implementation and software implementation

mented on the Virtex device XCV400HQ240. The Virtex user programmable gate array (vide Appendix B) comprises three major configurable elements, namely configurable logic blocks (CLBs), block RAMs (BRAMs) and input/output blocks (IOBs). The CLBs are the primary logic elements, which provides the functional elements for computation of logic operation. Each CLB is made of two slices, which contain two LUTs and two D flip-flops. The Virtex series has a system of block RAMs, which allows the use of the chip for limited RAM operations such as FIFO implementation or basic

dual-port RAM usage. The IOBs provide the interface between the device pins and the CLBs. The Virtex architecture also includes the delay-locked loop (DLL) to eliminate skew between the clock input pad and the internal clock-input pins throughout the device.

FPGA design summary of the entire architecture and the major functional blocks is shown in Table 5.4. The performance of the FPGA implementation is limited to 48.52 MHz in XCV400-6PQ240. The design utilizes almost 52% of CLBs and 25% of Registers.

Hardware Elements	Usage
Number of Slices	2503 out of 4800 (52%)
Slice Registers	2481 out of 9,600 (25%)
Block RAMs	6 out of 20 (30%)
IOBs	36 out of 166 (22%)
GCLKs	1 out of 4 (25%)
Total equivalent gate count: 102,546	
Maximum frequency of operation of entire design: 48. 52 MHz	
Maximum frequency of operation of FIFO Queue: 139. 76 MHz	
Maximum frequency of operation of NAG Block: 26. 92 MHz	
Maximum frequency of operation of Plateau Detection Block: 81. 05 MHz	
Maximum frequency of operation of Plateau Analysis Block : 58. 35 MHz	
Maximum frequency of operation of Lower Complete Block : 62.33 MHz	
Maximum frequency of operation of HCD Block : 48.43 MHz	
Maximum frequency of operation of CNC Block : 44. 56 MHz	

Table 5.4: FPGA device utilization summary

5.6 Conclusions

The present chapter proposes an improved watershed transform method based on simulated immersion and its prototype architecture. The proposed method is free of several disadvantages encountered by a conventional algorithm originally proposed by Vincent and Soillé [VS91]. The flooding mechanism in the proposed algorithm starts from pre-determined regional minima and is based on conditional neighborhood comparisons while dealing with the pixels surrounding a labeled center pixel. As a result, it always gives rise to thin continuous watershed lines as opposed to thick discontinuous lines which generally result from applying Vincent-Soillé algorithm. The proposed algorithm also incorporates the ability to decide labels of eight neighboring pixels at the same time. It demonstrates reduced computational complexity and increased throughput compared to the conventional algorithm. The complexity

of the proposed algorithm has been analyzed. Segmentation of various images produced by application of the proposed method and the conventional one are presented in this chapter for comparison. A prototype architecture for the proposed algorithm has been implemented in Xilinx FPGA. As the Xilinx Virtex device has limited on-chip memory, this design has been verified for an image of size 30×30. The FPGA implementation results too are given in this chapter.

Chapter 6

Conclusions and Future Work

In this final chapter, we present a summary of the chief contributions of the work reported in depth in the previous chapters of this thesis. Moreover, we point out a few areas that may be explored for further progress of the present research.

6.1 Summary of Work Done

The major part of the work reported in this thesis has been formulation of serial watershed segmentation algorithms that have evolved by overcoming the drawbacks experienced by a few conventional algorithms. The advantages of the proposed improved algorithms over the conventional algorithms have been demonstrated on various standard images like Table Tennis, Lena, Cermet, Washington_ir, MRI Brain, Head, Blood Cells, Steel Fissure, Akiyo, Claire and Peppers. Moreover, the present work has come up with appropriate hardware architectures to realize the proposed algorithms.

Chapter 1 provides the general motivation behind the present work. A review of the existing work on watershed algorithm also finds place here. Chapter 2 provides a systematic study of the multi-scale morphological gradient operator which is much used to prevent undesired over-segmentation that can occur in course of image segmentation by the watershed algorithm. Chapter 3 focusses on developing an improved version of a conventional flooding-based watershed algorithm [BM93] based on ordered queues, and a prototype architecture for realizing the same. Chapter 4 then identifies the scope of increasing the efficiency of a hillclimbing-based watershed algorithm, and based on this, proposes a new watershed algorithm with reduced computational complexity. Besides, an architecture involving moderate hardware complexity has been developed to implement the proposed hillclimbing-based algorithm. Chapter 5 points out at drawbacks inherent in a conventional immersion-based watershed

computation method, and comes up with an improved, efficient immersion-based watershed algorithm as well as its prototype architecture.

The principal points of the work reported in this dissertation may be summed up as follows:

- An improved flooding-based watershed algorithm is proposed, which constructs 0-width watersheds. The proposed simulated flooding technique uses a single queue as opposed to 256 queues needed in a conventional watershed algorithm [BM93]. Moreover, the proposed method can decide the labels of the eight surrounding neighbors of a labeled center pixel in parallel based on conditional neighborhood comparisons. This algorithm exhibits an equivalent performance at reduced hardware complexity while compared to the conventional watershed algorithm [BM93]. The homogeneous regions are identified by both the algorithms in an almost identical manner.

- Apart from demonstrating its satisfactory performance on a number of images, we have developed a prototype hardware architecture for an effective realization of the proposed flooding-based watershed algorithm. The architecture has been synthesized and implemented in an appropriate FPGA environment. The implementation results show acceptable performance of the proposed architecture. The proposed flooding-based algorithm and its prototype architecture have been reported in [CIA04, RCM03, RCM02].

- The drawback of the proposed flooding-based algorithm mentioned above is that no synchronization in label propagation is possible in a non-minimum plateau which has more than one closely located regional minima. It leads to a consequent reduction in the performance measure. To overcome it, Moga's method [Mog97] can be employed; but it is costly (computationally expensive). A fast hillclombing-based watershed algorithm, which constructs 0-width watershed lines by avoiding time-consuming lower complete transformation is introduced. Implementation of Moga's algorithm requires considerably more hardware overhead, because it requires a lower complete image to be computed that would require a multiplier unit. Instead of that, only two more comparisons have been included in the flooding process of the proposed algorithm. To reduce the overall computation further, some other modification has been adopted, namely employment of two FIFO queues and computation of the steepest lower neighbor image during the process of detection of minima/non-minima plateau. The ordering relation necessary during the flooding procedure (or label propagation) has been theoretically established. This technique requires only about $3n$ comparisons as against $3n$ comparisons, n multiplications and n additions

for the conventional algorithm [Mog97], where n is the dimension of the image. The reduced complexity makes the algorithm suitable for hardware implementation. The segmentation results show that the homogeneous regions are identified by both algorithms in an identical manner.

- A prototype architecture has also been developed for an effective realization of the proposed hillclimbing technique. The architecture has been synthesized and developed in an appropriate FPGA environment. Adopting a strategy of sharing of resources such as FIFOs, counters, muxes and demuxes between the two major hardware blocks and employing a single FSM for the control unit, this design has incurred even less hardware cost compared to the architecture required to implement the proposed improved flooding-based watershed algorithm. The proposed hillclimbing technique and its prototype architecture have been reported in [RRC03].

- An improved watershed transform method based on simulated immersion is proposed, which constructs watershed lines of uniform width. The proposed method is free of several disadvantages encountered by a conventional algorithm originally proposed by Vincent and Soillé [VS91]. The flooding mechanism in the proposed algorithm starts from pre-determined regional minima and is based on conditional neighborhood comparisons while dealing with the pixels surrounding a labeled center pixel. In all, four possible different cases have been identified to occur for deciding the labels of the 8-connected un-labeled neighbors of a labeled center pixel. As a result, the proposed method always gives rise to thin continuous watershed lines as opposed to thick discontinuous lines which generally result from applying the algorithm due to Vincent and Soillé.

- The proposed immersion-based watershed algorithm also derives its strength from the parallelization of the label propagation process. The conditional neighborhood comparison in label propagation procedure helps one decide the labels of eight neighboring pixels at the same time. It demonstrates reduced computational complexity and increased throughput compared to the conventional algorithm [VS91]. The performance of the label propagation is improved from $\frac{1}{9}$=11% to $\frac{8}{25}$ = 32%. The proposed immersion-based watershed algorithm requires $4n$ comparisons as against $5n$ comparisons for the conventional algorithm [VS91], where n is the dimension of the image. The proposed immersion-based watershed algorithm has been reported in [RC03, RCG03].

- A 2-D cellular automata (CA)-based prototype architecture has been proposed for an effective

realization of the proposed algorithm. A fully parallel synchronous (3×3) 2-D CA model with eight processing elements (cells) for eight neighbors of the center pixel is used in accomplishing the conditional neighborhood comparisons. This architecture, which demonstrates high speed of operation due to local interconnection, has been synthesized and implemented in an appropriate FPGA environment.

- Superiority of the proposed algorithms over the conventional algorithms has been conclusively demonstrated on various standard images. The prototype architectures for the proposed algorithms have been implemented in Xilinx FPGA. As the Xilinx Virtex device has limited on-chip memory, the hardware designs have been verified for an image of size 30×30.

6.2 Future Work

The present section indicates a few areas that may be explored by extending the scope of the present work.

- ☐ **Multiresolution-based watershed algorithm for image segmentation:** There are two main drawbacks when applying the watershed algorithm to image segmentation: possibility of over-segmentation and high computational complexity to merge the over-segmented regions. These problems can be eliminated when the watershed algorithm is integrated within a multiresolution approach [MPL00, WLGW01, KK03]. This approach has a filtering operation which reduces the image noise and the computational complexity. The watershed algorithms proposed in this thesis can be improved further by multiresolution approach, in which the watershed operation is carried out on the low-pass filtered low-resolution images resulting from the wavelet transform. Hence, the computational complexity can be simplified and reduced dramatically by operating on a low-resolution image. After image segmentation, the segmented labelled low-resolution image is projected at full-resolution image by an inverse wavelet transform. This method can be used for extracting video objects from image sequences.

- ☐ **Watershed transform on a Reconfigurable Multi-Ring Network (RMRN):** The RMRN is a scalable network in which each node has a fixed degree of connectivity and reconfiguration mechanism that ensures a network diameter $O(\log_2 N)$ for an N-processor network. The algorithms capable of being implemented on the SPMD/SIMD architecture can be mapped to the RMRN structure. Various RMRN architectures [BA95, BA97, WA03] have been proposed for complex

image processing and computer vision problems. The performance of the proposed watershed algorithms can be improved by parallel computation in a reconfigurable multi-ring network of processors. An image can be divided into N blocks corresponding to N processors. Each processor computes the watershed transform in its own domain. Further communication amongst the processors is required to assign the same label to a plateau which is shared by different processors. One can attempt to develop dedicated hardware architecture based on RMRN to realize the parallel watershed transform computation as well.

- **Dynamically reconfigurable architecture:** Over the last 10 years, configurable computing machines [Deh00] have demonstrated their efficiency to execute complex signal and image processing algorithms for several applications like convolution, morphology, image filtering, edge extraction and object recognition. The most common devices used for configurable computing are field programmable gate arrays (FPGA). The FPGAs have demonstrated the potential for achieving configurable computing and prototype systems for a wide domain of applications. The majority of the FPGA structures have a fine-grained architecture. FPGAs are composed of configurable logic blocks which are used as custom hardware capable of being configured to implement the required function. These hardware resources can be used to design custom computing machines highly adapted to a specific application. If the logic resources are targeted to execute one application, the chip is configured only at the startup time and remains unchanged until the application is finished. Data-level parallelism can be adopted to increase the performance of an FPGA-based hardware design. It is necessary to reconfigure the logic functionality and set connectivity at bit level. FPGA reconfiguration typically requires the whole chip to be reprogrammed even for the slightest circuit change. If the configuration interface is slow, then the configuration time can be a significant overhead in configurable systems, which limits the performance when an FPGA needs to be reconfigured. Alternatively, dynamically reconfigurable systems can perform various computational tasks through hardware reuse. This approach can reduce the hardware complexity and increase the functional density of the reconfigurable resources for sharing the same physical chip to perform all the stages in a particular application. Dynamic reconfiguration implies modifying the system when it is under operation. Various architectures that are dynamically reconfigurable have been proposed in the literature [WH98, KAD03, EG03].

  A dynamically reconfigurable architecture to realize the proposed improved watershed algorithms may be developed. Dynamic reconfiguration offers the possibility of implementing on a

single FPGA device, the major algorithmic steps of local minima detection and label propagation in a time-multiplexed fashion.

Appendix A

VHDL Design of the (3×3) 2-D CA Model for Conditional Neighborhood Comparison Process

The VHSIC Hardware Description Language (VHDL) [vhd93, Bha99, Per01] is an industrial standard language used to model a digital system at many levels of abstraction, ranging from the architecture level to the gate level. It offers a broad set of constructs for describing even the most complicated digital system in a compact fashion. The digital systems can be described hierarchically. The structural descriptions assemble several blocks and allow the introduction of hierarchy in a design. Timing can also be explicitly modelled in the same description. In VHDL, there are two types of delay that can be used for modelling behaviors. Inertial delay is most commonly used, while transport delay is used where a wire delay model is required.

A circuit description consists of two parts: the interface part and the body. In VHDL, the *entity* statement corresponds to the interface and the *architecture* statement describes the behavior. A digital system is modelled using an entity declaration and at least one architecture body. The I/O ports of the circuit are declared in the entity. Each port has a name, a mode and a type. The architecture body contains the internal description of the entity; for example, as a set of interconnected components that represent the structure of the entity or as a set concurrent or sequential statements that represent the behavior of the entity. A single entity can have multiple architectures.

A *process* statement contains one or more sequential statements that describe the functionality of a portion of an entity in sequential terms. The process statement consists of three parts, namely the sensitivity list, the process declaration and the statement part. The sensitivity list enumerates exactly

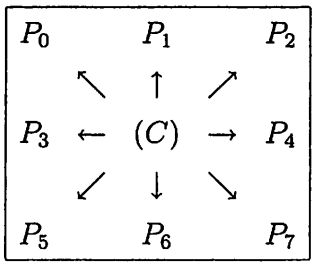
which signals would cause the process statement to be executed. The process declaration part consists of the area between the end of the sensitivity list and the keyword 'begin'. This area is used to declare local variables or constants that can be used only inside the process. The statement part of the process starts at the keyword 'begin' and ends at the 'end process' line. All of the statements enclosed by the process are sequential statements. This means that any statements enclosed by the process are executed one after another in a sequential order. The order of execution is the order of the statements in the process statement. A variable assignment statement assigns a value to a variable using the operator ":=" and a signal assignment statement assigns a value to a signal using "<=".

A *package* is a collection of commonly used data types, function, procedures and components used in a design. It consists of two sections, namely the package declaration and the package body. The package body declaration includes the description of the function blocks declared in the package declaration. The declaration of a *function* or a *procedure* provides a mechanism for handling blocks used many times in a design. Functions and procedures can be declared in the declarative part of an entity in an architecture, or in packages. A comprehensive description of various constructs and features of VHDL is found in [vhd93, Bha99, Per01].

The prototype architectures for the improved watershed algorithms introduced in Chapter 3-5 have been modelled in VHDL. A sample VHDL code for (3 × 3) 2-D cellular automata (CA) model for conditional neighborhood comparison process is given next.

A.1 VHDL Code for (3 × 3) 2-D CA Model

The (3 × 3) 2-D CA Model accommodates the eight cells (PE) P_0, P_1, ..., P_7 (vide Figure 5.22 of Chapter 5) involved in comparing the intensities as well as the labels of the neighboring pixels (from adjacent cells) and supporting pixels with the center pixel intensity value (b_data) and label value (b_label). The 8-connected neighborhood processing elements of the center pixel C is given by



The component cond_neigh_comp used in the conditional neighborhood comparison process may be described by means of VHDL code as follows.

```
– – – – – – – – – – – – – – – – – cond_neigh_comp.vhd – – – – – – – – – – – – – – –
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library work;
use work.ram_pkg.all;
– – – Entity declaration – – – –
entity cond_neigh_comp is
    Port (reset, clk, logic0, logic1: in std_logic;
        logic_vec1, NARM, B_data, B_label, B_dist : in  std_logic_vector(7 downto 0);
        NS_DATA, NS_LABEL : in m_type_24;
        NS_DIST : in m_type_8;
        r_s_flag : in std_logic_vector(23 downto 0);
        l_w_flag, f_w_flag : out std_logic_vector(7 downto 0);
        complete_flag : out std_logic );
end cond_neigh_comp;
– – – Architecture declaration – – – –
architecture cond_neigh_comp_arch of cond_neigh_comp is
signal comp_flag : std_logic_vector(7 downto 0);
begin
– – – Top left neighbor of the center pixel – – – –
P0: PE port map(reset => reset, clk => clk, logic0 => logic0, logic1 => logic1, logic_vec1 => logic_vec1,
        narm => narm, b_data => B_data, B_label => B_label, B_dist => B_dist, n_r_flag => r_s_flag(0),
        n_data => NS_DATA(0), n_label => NS_LABEL(0), n_dist => NS_DIST(0), n_r_flag_1 => r_s_flag(8),
        n_r_flag_2 => r_s_flag(9), n_r_flag_3 => r_s_flag(10), n_r_flag_4 => r_s_flag(13), n_r_flag_5 => r_s_flag(1),
        n_r_flag_6 => r_s_flag(15), n_r_flag_7 => r_s_flag(3), n_data_1 => NS_DATA(8),
        n_data_2 => NS_DATA(9), n_data_3 => NS_DATA(10), n_data_4 => NS_DATA(13),
        n_data_5 => NS_DATA(1), n_data_6 => NS_DATA(15), n_data_7 => NS_DATA(3),
        n_label_1 => NS_LABEL(8), n_label_2 => NS_LABEL(9), n_label_3 => NS_LABEL(10),
        n_label_4 => NS_LABEL(13), n_label_5 => NS_LABEL(1), n_label_6 => NS_LABEL(15),
        n_label_7 => NS_LABEL(3), l_w_flag => l_w_flag(0), f_w_flag => f_w_flag(0),
        complete_flag => comp_flag(0));
-- Next, the port map statements for P1, P2, P3, P4, P5 and P6 are to be written. They are however, not given
in the present thesis for paucity of space --
– – – Top left neighbor of the center pixel – – – –
P1: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(1));
```

```
– – – Top neighbor of the center pixel – – ––
P2: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(2));
– – – Left neighbor of the center pixel – – ––
P3: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(3));
– – – Right neighbor of the center pixel – – ––
P4: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(4));
– – – Bottom left neighbor of the center pixel – – ––
P5: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(5));
– – – Bottom neighbor of the center pixel – – ––
P6: PE port map(reset => reset, clk => clk, . . .,    , . . .,    , complete_flag => comp_flag(6));
– – – Bottom right neighbor of the center pixel – – ––
P7: PE port map(reset => reset, clk => clk, logic0 => logic0, logic1 => logic1, logic_vec1 => logic_vec1,
      narm => narm, b_data => B_data, B_label => B_label, B_dist => B_dist, n_r_flag => r_s_flag(7),
      n_data => NS_DATA(7), n_label => NS_LABEL(7), n_dist => NS_DIST(7), n_r_flag_1 => r_s_flag(4),
      n_r_flag_2 => r_s_flag(16), n_r_flag_3 => r_s_flag(6), n_r_flag_4 => r_s_flag(18), n_r_flag_5 => r_s_flag(21),
      n_r_flag_6 => r_s_flag(22), n_r_flag_7 => r_s_flag(23), n_data_1 => NS_DATA(4),
      n_data_2 => NS_DATA(16), n_data_3 => NS_DATA(6), n_data_4 => NS_DATA(18),
      n_data_5 => NS_DATA(21), n_data_6 => NS_DATA(22), n_data_7 => NS_DATA(23),
      n_label_1 => NS_LABEL(4), n_label_2 => NS_LABEL(16), n_label_3 => NS_LABEL(6),
      n_label_4 => NS_LABEL(18), n_label_5 => NS_LABEL(21), n_label_6 => NS_LABEL(22),
      n_label_7 => NS_LABEL(23), l_w_flag => l_w_flag(7), f_w_flag => f_w_flag(7),
      complete_flag => comp_flag(7));
–– A status signal complete_flag is generated when all the processing elements are processed ––
complete_flag <= comp_flag(0) and comp_flag(1) and comp_flag(2) and comp_flag(3) and comp_flag(4)
                 and comp_flag(5) and comp_flag(6) and comp_flag(7);
end cond_neigh_comp_arch;
– – – – – – – – – – End of the component cond_neigh_comp – – – – – – – – – – – – – –
– – – – – – – – – – – – – – – – ram_pkg.vhd – – – – – – – – – – – – – – – – – – –
–– This package defines the supplemental types, subtypes, constants and functions ––
package ram_pkg is
type m_type is array (255 downto 0) of std_logic_vector(7 downto 0);
type m_type_32 is array (31 downto 0) of std_logic_vector(7 downto 0);
type m_type_24 is array (23 downto 0) of std_logic_vector(7 downto 0);
type m_type_8 is array(7 downto 0) of std_logic_vector(7 downto 0);
end ram_pkg;
– – – – – – – – – – End of the package ram_pkg – – – – – – – – – – – –
```

Component PE is instantiated in the component cond_neigh_comp may be described as follows.

```
- - - - - - - - - - - - - - - - - - - - - - PE.vhd - - - - - - - - - - - - - - - - - - - - - - -
entity PE is
    Port (reset, clk, logic0,logic1, n_r_flag : in std_logic;
        logic_vec1, NARM, b_data, b_label, b_dist : in std_logic_vector(7 downto 0);
        n_data, n_label, n_dist : in std_logic_vector(7 downto 0);
        n_r_flag_1,n_r_flag_2,n_r_flag_3,n_r_flag_4 : in std_logic;
        n_r_flag_5,n_r_flag_6,n_r_flag_7 : in std_logic;
        n_data_1,n_data_2,n_data_3,n_data_4,n_data_5,n_data_6 : in std_logic_vector(7 downto 0);
        n_r_flag_7 , n_label_1,n_label_2,n_label_3 : in std_logic_vector(7 downto 0);
        n_label_4,n_label_5,n_label_6,n_label_7 : in std_logic_vector(7 downto 0);
        l_w_flag, f_w_flag,complete_flag : out std_logic );
end PE;
-- Architecture of the PE entity --
architecture PE_Arch of PE is
-- signal declaration --
signal b_n_comp_flag, n_l_comp_flag_1, n_l_comp_flag_2, n_l_comp_flag_3 : std_logic;
signal n_l_comp_flag_4, n_l_comp_flag_5, n_l_comp_flag_6, n_l_comp_flag_7 : std_logic;
signal nr_comp_flag_1, nr_comp_flag_2, nr_comp_flag_3, nr_comp_flag_4 : std_logic;
signal nr_comp_flag_5, nr_comp_flag_6, nr_comp_flag_7 : std_logic;
signal n_l_comp,nr_comp,l_r_comp, b_n_eq_flag, c_flag : std_logic;
begin
-- The B_N module is used to detect the validity of the deciding neighboring pixel n_data --
b_n_block: B_N port map(reset => reset, clk => clk, logic0 => logic0, logic1 => logic1,
                logic_vec1 =>logic_vec1, narm => narm, n_r_flag => n_r_flag, B_data => B_data,
                B_dist => B_dist, N_data => N_data, N_label => N_label, N_dist => N_dist,
                comp_flag => b_n_comp_flag, b_n_eq_flag => b_n_eq_flag );
-- The components n1, n2, ..., n7 are used to decide the label of the neighboring pixel n_data based on four
conditions --
n1: n_n port map(reset => reset, clk => clk, logic0 => logic0, logic1 => logic1, narm => narm,
                n_r_flag => n_r_flag, n_r_flag_s => n_r_flag_1, n_data => n_data, N_data_s => N_data_1,
                B_label => B_label, N_label_s => N_label_1, n_d_comp_flag => n_l_comp_flag_1,
                nr_d_comp_flag => nr_comp_flag_1);
n2: n_n port map(reset => reset, clk => clk, logic0 => logic0, logic1 => logic1, narm => narm,
                n_r_flag => n_r_flag, n_r_flag_s => n_r_flag_2, n_data => n_data, N_data_s => N_data_2,
                B_label => B_label, N_label_s => N_label_2, n_d_comp_flag => n_l_comp_flag_2,
                nr_d_comp_flag => nr_comp_flag_2);
```

```
-- Similarly, the components n3, n4, n5, n6 and n7 can be instantiated --
-- The n_l_comp signal is generated when one of the four conditions (vide figure 5.8) is satisfied --
n_l_comp <= n_l_comp_flag_1 and n_l_comp_flag_2 and n_l_comp_flag_3 and n_l_comp_flag_4
            and n_l_comp_flag_5 and n_l_comp_flag_6 and n_l_comp_flag_7;
-- The nr_comp status signal is generated when all n1, n2, ..., n7 modules are processed --
nr_comp <= nr_comp_flag_1 and nr_comp_flag_2 and nr_comp_flag_3 and nr_comp_flag_4
            and nr_comp_flag_5 and nr_comp_flag_6 and nr_comp_flag_7;
-- The component m_2x1 used for signal selection --
m1: m_2x1 port map(data_in1 => n_l_comp, data_in2 => logic0, sel => nr_comp, data_out => l_r_comp);
m2: m_2x1 port map(data_in1 => logic0, data_in2 => l_r_comp, sel => b_n_comp_flag, data_out => l_w_flag);
m3: m_2x1 port map(data_in1 => logic0, data_in2 => b_n_eq_flag, sel => b_n_comp_flag, data_out =>
f_w_flag);
m4: m_2x1 port map(data_in1 => c_flag, data_in2 => logic0, sel => reset, data_out => complete_flag);
m5: m_2x1 port map(data_in1 => logic1, data_in2 => logic1, sel => b_n_comp_flag, data_out => c_flag);
end PE_Arch;
- - - - - - - - - - - - - - - - - - End of the component PE - - - - - - - - - - - - - - - - - -
```

The components B_N and n_n, which are instantiated in the component PE may be given in VHDL code as follows.

```
- - - - - - - - - - - - - - - - - - - - - - B_N.vhd - - - - - - - - - - - - - - - - - - - - - - -
entity B_N is
    Port (reset, clk, logic0,logic1, n_r_flag : in std_logic;
       logic_vec1, narm, B_data, B_dist, N_data, N_label, N_dist : in std_logic_vector(7 downto 0);
       comp_flag, b_n_eq_flag : out std_logic );
end B_N;
-- Architecture of the B_N entity --
architecture Behavioral of B_N is
-- signal declaration --
signal N_label_comp,n_sum,n_data_out,n_dist_out,s_d : std_logic_vector(7 downto 0);
signal N_l_c_flag,N_L_comp_flag,N_D_comp_flag,compare_flag : std_logic;
signal n_s_flag,dist_comp : std_logic;
signal g_eq_comp,g_eq_flag,eq_flag : std_logic;
signal b_n_comp : std_logic;
begin
r1: reg_syn port map(reset => reset, load => n_r_flag, clk => clk,
                     data_in => N_label, data_out => N_label_comp);
```

```
-- compare the n_label with NARM --
c1: compare_eq port map(A => N_label_comp, B => NARM, C => N_l_c_flag);
m1: m_2x1 port map(data_in1 => logic0, data_in2 => N_l_c_flag, sel => n_r_flag, data_out => N_L_comp_flag);
-- compare the n_data with b_data --
r2: reg_syn port map(reset => reset, load => n_r_flag, clk => clk, data_in => n_data, data_out => n_data_out);
am1: AminusB port map(A => n_data_out, B => B_data, Cin => logic1, S => n_sum, sign_flag => n_s_flag);
g_eq_flag <= n_sum(7) or n_sum(6) or n_sum(5) or n_sum(4) or n_sum(3) or n_sum(2) or n_sum(1) or n_sum(0);
-- compare the n_dist with B_dist --
ap1: AplusB port map(A => B_dist, B => logic_vec1, Cin => logic0, S => s_d);
r3: reg_syn port map(reset => reset, load => n_r_flag, clk => clk, data_in => N_dist, data_out => N_dist_out);
c2: compare_eq port map(A => s_d, B => N_dist_out, C => dist_comp);
m2: m_2x1 port map(data_in1 => dist_comp, data_in2 => logic1, sel => g_eq_flag, data_out => eq_flag);
m3: m_2x1 port map(data_in1 => logic0, data_in2 => eq_flag, sel => n_s_flag, data_out => N_D_comp_flag);
compare_flag <= N_l_comp_flag and N_D_comp_flag;
m4: m_2x1 port map(data_in1 => logic0, data_in2 => compare_flag, sel => n_r_flag, data_out => comp_flag);
m5: m_2x1 port map(data_in1 => dist_comp, data_in2 => logic0, sel => g_eq_flag, data_out => b_n_comp);
-- The b_n_eq_flag status signal is generated when the neighboring pixel n_data greater than the center pixel
b_data, the distance n_dist equal to one or more than the center pixel distance B_dist and the neighboring pixel
label n_label is equal to NARM --
b_n_eq_flag <= b_n_comp and n_l_comp_flag;
end Behavioral;
- - - - - - - - - - - - - - - - - - End of the component B_N - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - n_n.vhd - - - - - - - - - - - - - - - - - - - - - - -
entity n_n is
    Port (reset, clk, logic0, logic1, n_r_flag, n_r_flag_s : in std_logic;
        narm, N_data, N_data_s, B_label, N_label_s : in std_logic_vector(7 downto 0);
        n_d_comp_flag, nr_d_comp_flag : out std_logic );
end n_n;
-- Architecture of the n_n entity --
architecture Behavioral of n_n is
-- signal declaration --
signal n_data_out,n_data_s_out,s_d,n_label_s_out : std_logic_vector(7 downto 0);
signal n_s_flag,eq_flag,g_eq_flag : std_logic;
signal n_l_c_flag, n_l_n_flag,n_l_comp,n_l_comp_flag : std_logic;
signal nr_comp_flag : std_logic;
begin
r1: reg_syn port map(reset => reset, load => n_r_flag_s, clk => clk,
```

```
                    data_in => N_data_s, data_out => n_data_s_out);
r2: reg_syn port map(reset => reset, load => n_r_flag, clk => clk, data_in => N_data, data_out => n_data_out);
-- compare n_data with the supporting pixel n_data_s --
c1: AminusB port map(A => N_data_s_out, B => N_data_out, Cin => logic1, S => S_D, sign_flag =>
n_s_flag);
eq_flag <= s_d(7) or s_d(6) or s_d(5) or s_d(4) or s_d(3) or s_d(2) or s_d(1) or s_d(0);
g_eq_flag ¡= n_s_flag xor eq_flag;
r3: reg_syn port map(reset => reset, load => n_r_flag_s, clk => clk, data_in => N_label_s, data_out =>
n_label_s_out);
-- compare n_label with the n_label_s --
c2: compare_eq port map(A => N_label_s_out, B => B_label, C => n_l_c_flag);
c3: compare_eq port map(A => N_label_s_out, B => NARM, C => n_l_n_flag);
m1: m_2x1 port map(data_in1 => n_l_c_flag, data_in2 => logic1,
                    sel => n_l_n_flag, data_out => n_l_comp);
m2: m_2x1 port map(data_in1 => logic1, data_in2 => n_l_comp,
                    sel => g_eq_flag, data_out => n_l_comp_flag);
-- The nr_comp_flag status signal is generated when one of the four conditions (vide figure 5.8) is satisfied --
nr_comp_flag <= not(g_eq_flag) xor n_l_n_flag;
m3: m_2x1 port map(data_in1 => logic1, data_in2 => n_l_comp_flag,
                    sel => n_r_flag_s, data_out => n_d_comp_flag);
m4: m_2x1 port map(data_in1 => logic0, data_in2 => nr_comp_flag,
                    sel => n_r_flag_s, data_out => nr_d_comp_flag);
end Behavioral;
- - - - - - - - - - - - - - - - - - End of the component n_n - - - - - - - - - - - - - - - - - - -
```

The components compare_eq, AminusB, reg_syn and m_2x1, which are instantiated in the above modules may be described as follows.

```
    - - - - - - - - - - - - - - - - - - compare_eq.vhd - - - - - - - - - - - - - - - - - - -
entity compare_eq is
    Port (A, B : in std_logic_vector(7 downto 0);
        C : out std_logic);
end compare_eq;
architecture Behavioral of compare_eq is
signal temp : std_logic_vector(7 downto 0);
begin
```

```
temp <= A xor B;
C <= not(temp(7) or temp(6) or temp(5) or temp(4) or temp(3) or temp(2) or temp(1) or temp(0));
end Behavioral;
- - - - - - - - - - - - - - - - - End of the component compare_eq - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - AminusB.vhd - - - - - - - - - - - - - - - - - -
entity AminusB is
    Port ( A, B : in std_logic_vector(7 downto 0);
        Cin : in std_logic;
        S : out std_logic_vector(7 downto 0);
        sign_flag : out std_logic);
end AminusB;
architecture Behavioral of AminusB is
signal cg, cp,B_M : std_logic_vector(7 downto 0);
signal C : std_logic_vector(7 downto 0);
begin
cg <= A and (not(B));
cp <= A xor (not(B)); C(0) ¡= Cin;
g1: for i in 0 TO 7 generate
S(i) <= cp(i) XOR C(i);
end generate;
C(1) <= cg(0) or cp(0);
C(2) <= cg(1) or (cp(1) and cg(0)) or (cp(1) and cp(0));
C(3) <= cg(2) or (cp(2) and cg(1)) or (cp(2) and cp(1) and cg(0)) or (cp(2) and cp(1) and cp(0));
C(4) <= cg(3) or (cp(3) and cg(2)) or (cp(3) and cp(2) and cg(1)) or (cp(3) and cp(2) and cp(1) and cg(0)) or
(cp(3) and cp(2) and cp(1) and cp(0));
C(5) ¡= cg(4) or (cp(4) and cg(3)) or (cp(4) and cp(3) and cg(2)) or (cp(4) and cp(3) and cp(2) and cg(1)) or
(cp(4) and cp(3) and cp(2) and cp(1) and cg(0)) or (cp(4) and cp(3) and cp(2) and cp(1) and cp(0));
C(6) ¡= cg(5) or (cp(5) and cg(4)) or (cp(5) and cp(4) and cg(3)) or (cp(5) and cp(4) and cp(3) and cg(2)) or
(cp(5) and cp(4) and cp(3) and cp(2) and cg(1)) or (cp(5) and cp(4) and cp(3) and cp(2) and cp(1) and cg(0))
or (cp(5) and cp(4) and cp(3) and cp(2) and cp(1) and cp(0));
C(7) ¡= cg(6) or (cp(6) and cg(5)) or (cp(6) and cp(5) and cg(4)) or (cp(6) and cp(5) and cp(4) and cg(3)) or
(cp(6) and cp(5) and cp(4) and cp(3) and cg(2)) or (cp(6) and cp(5) and cp(4) and cp(3) and cp(2) and cg(1))
or (cp(6) and cp(5) and cp(4) and cp(3) and cp(2) and cp(1) and cg(0)) or (cp(6) and cp(5) and cp(4) and
cp(3) and cp(2) and cp(1) and cp(0));
sign_flag ¡= cg(7) or (cp(7) and cg(6)) or (cp(7) and cp(6) and cg(5)) or (cp(7) and cp(6) and cp(5) and cg(4))
or (cp(7) and cp(6) and cp(5) and cp(4) and cg(3)) or (cp(7) and cp(6) and cp(5) and cp(4) and cp(3) and
cg(2)) or (cp(7) and cp(6) and cp(5) and cp(4) and cp(3) and cp(2) and cg(1)) or (cp(7) and cp(6) and cp(5)
```

```
and cp(4) and cp(3) and cp(2) and cp(1) and cg(0)) or (cp(7) and cp(6) and cp(5) and cp(4) and cp(3) and
cp(2) and cp(1) and cp(0));
end Behavioral;
- - - - - - - - - - - - - - - - - - End of the component AminusB - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - reg_syn.vhd - - - - - - - - - - - - - - - - - - -
entity reg_syn is
    Port ( reset, load, clk : in std_logic;
    data_in : in std_logic_vector(7 downto 0);
    data_out : out std_logic_vector(7 downto 0));
end reg_syn;
architecture Behavioral of reg_syn is
begin
process(clk,reset,load)
begin
    if clk'event and clk='1' then
        if reset = '0' and load = '1' then
            data_out <= data_in;
        end if;
    end if;
end process;
end Behavioral;
- - - - - - - - - - - - - - - - - - End of the component reg_syn - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - m_2x1.vhd - - - - - - - - - - - - - - - - - - -
entity m_2x1 is
    Port ( data_in1, data_in2, sel : in std_logic;
        data_out : out std_logic); end m_2x1;
architecture Behavioral of m_2x1 is
begin
process(data_in1,data_in2,sel)
begin
    case sel is
        when '0' =>
            data_out <= data_in1;
        when '1' =>
            data_out <= data_in2;
        when others => NULL;
    end case;
```

end process;

end Behavioral;

– – – – – – – – – – – – – – – – – – End of the component m_2x1 – – – – – – – – – – – – – – – – – – –

– – – – End of the VHDL specification of conditional neighborhood comparison block – –

Appendix B

Xilinx FPGA

Programmable devices such as Programmable Array Logic (PAL) or Programmable Logic Devices (PLD), mask-programmed Gate Arrays or Cell-based ASICs, and Field Programmable Gate Arrays (FPGAs) [Tri94] are widely used in industry. Of these three broad classes, the programmable devices are usually smaller in capacity but more predictable in timing, and they can be implemented in sum-of-products form , product-of-sums form or both. A simple PLD is a general purpose device capable of implementing the logic of tens or hundreds of Small-Scale Integrated (SSI) packages. It can be programmed by users at their site using inexpensive hardware. Power consumption and delay can limit the size of the simple sum-of-products structure to a few dozens of products terms. Large designs require a multi-level logic implementation. To implement designs with thousands or tens of thousands of gates, designers can use a Mask-Programmable Gate Array (MPGA). A typical MPGA can implement tens of thousands or even hundreds of thousands of gates of logic on a single IC in multi-level logic wiring between logic stages. Typical turnaround time for MPGA is four to six weeks.

Field programmable gate arrays offer the benefits of both programmable devices and gate arrays. The FPGAs can be implemented with thousands of gates of logic in a single chip and can be programmed by designers at their site, which eliminate the long delays and tool costs. Moreover, an FPGA can be programmed in minutes by users. On an FPGA, a modification to correct a design flaw or to address a late specification change can be made quickly and cheaply. It offers low risk, low incremental cost and fast prototyping advantages. The disadvantage of the FPGAs happens to be the on-chip programming overhead circuitry that manages the programming of the devices. The area occupied by the programming overhead circuitry cannot be used by users, and hence it lowers the FPGA gate density. Moreover, the programmable switches and options in an FPGA are larger than the mask programming. The programmable switches also increase the signal delay by adding resistance and capacitance to interconnect paths. Hence, the FPGAs are larger and slower than the equivalent MPGAs.

FPGA devices are based on Flash RAMs, SRAM (Static RAM), EEPROM or Anti-Fuse connectiv-

ities. The most successful FPGA devices are based on SRAM. This is because all other types are much less dense in terms of area than SRAM. Also, some types of connectivity is one-time programmable (Anti-Fuse); so, they are not very flexible. SRAM-based FPGAs have no maximum erase cycle limitation either. An SRAM-programmable FPGA is programmed by loading the configuration memory cells from an external source. The configuration memory cells control the logic and the interconnections that perform the application function of the FPGA. The disadvantage of volatility provides the advantage of reprogrammability. Reprogrammability makes the SRAM-programmable FPGAs ideal for prototype development. SRAM-based FPGAs include all three Xilinx families [xil] and Altera Flex [alt]. SRAM-based FPGAs have lookup tables which can be programmed to implement any function of N variables (usually $4 \sim 5$) and flip-flops which can be programmed to implement different types of storage (JK, T, Latch, DFF with set and/or reset).

Xilinx [x1] is vendor for FPGA devices. The Xilinx FPGA devices are organized as arrays of logic blocks surrounded by I/O blocks. The blocks are connected by programmable interconnects. Xilinx FPGAs use SRAM technology to control the interconnections. The logic block contains both combinational logic and registers. Xilinx markets SRAM-based devices in multi families including XC3000, XC4000, XC5200, Spartan and Virtex series. All devices are meant for general purpose. Any family can implement any type of logic. The maximum frequency of operation of a logic circuit and the FPGA components used for implementing the circuit depend on the chosen Xilinx device. The Virtex family delivers high-performance, high-capacity, programmable logic solutions among the Xilinx FPGA families. It supports the dedicated carry logic for high-speed arithmetic, a multiplier, the built-in clock management circuitry and the configurable synchronous dual-port block RAMs. For digital signal processing applications and image processing algorithms, this family is appropriate. A brief description of the architecture of a typical Virtex FPGA device is given next.

Virtex FPGA

The Virtex field programmable gate array, shown in Fig. B.1, comprises two major configurable elements: configurable logic blocks (CLBs) and input/output (IOBs). CLBs are the primary logic elements in the Virtex FPGA. It provides the functional elements for constructing logic. Each CLB is made of two slices as shown in Fig. B.2, each of which contains two Look Up Tables (LUTs) and two D flip-flops. Each LUT can be used as one 32×1-bit or one 16×2-bit synchronous RAM. The Virtex XCV100 has a 20×30 array of CLBs, resulting in a total of 2700 logic cells and 108, 904 gates. CLBs communicate through a general routing matrix (GRM). The GRM comprises an array of routing switches at the intersections of horizontal and vertical routing channels. Each CLB nests into a VersaBlock that also provides local routing resources to connect the CLB to the GRM. The VersaRing I/O interface provides

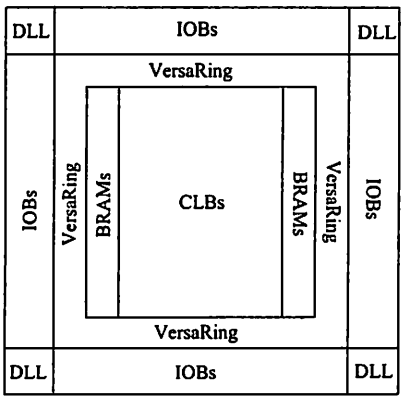


Figure B.1: Xilinx FPGA Virtex architecture.

additional routing resources around the periphery of the device.

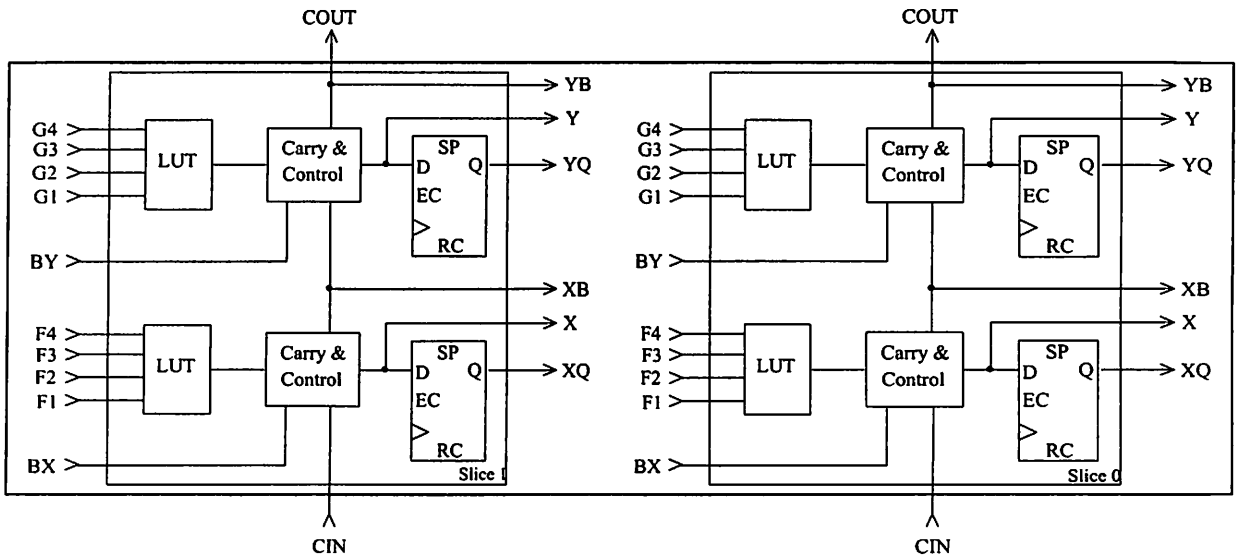


Figure B.2: A 2-slice Xilinx Virtex CLB.

IOBs provide the interface between the package pins and the CLBs. Input and output capabilities are handled by IOBs. The Virtex XCV100 has 180 IOBs. The Virtex series has a system of Block RAMs, each of which allows the use of the chip for limited RAM operations such as FIFO implementations [xil] or basic RAM usage. The XCV100 has 40,960 bits of Block RAM. The Virtex architecture also includes the delay-locked loops (DLLs) to eliminate skew between the clock input pad and the internal clock-input pins throughout the device.

Technically, the Xilinx FPGAs are SRAM devices. Values stored in static memory cells control

the configurable logic elements and the interconnect resources. These values are load into the memory cells on power-up, and reloading the values if necessary to change the function of the device.

Xilinx FPGA Design System

The design flow for Xilinx FPGA is shown in Figure B.3. FPGA development is in some sense similar to ASIC development. One can talk about the front-end tools (design entry tools) which can be schematic capture or an HDL (Hardware Description Language). Most common HDLs used for FPGA design are Verilog and VHDL. After describing an FPGA design in an HDL, a tool called a synthesizer effectively converts the HDL code into the specific primitives which can exist in an FPGA family. Most popular synthesis tools come from Synplicity, Exemplar, Cadence and Synopsis. Xilinx recommends

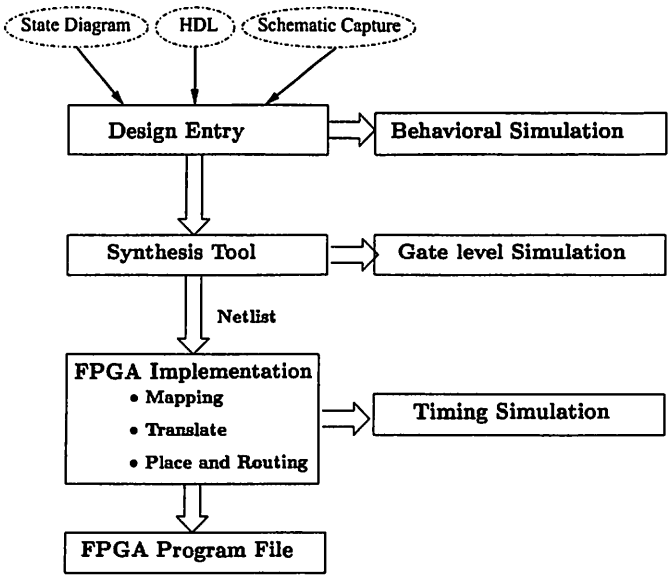


Figure B.3: Xilinx FPGA design system flow.

XST (Xilinx Synthesis Tool) and FPGA Express tool for synthesis. After a device level netlist is generated by synthesis, one uses a back-end tool called "place and route" which is supplied by the device vendor. From the place and route tool, one obtains a file which can be used to download onto an FPGA to program it with the hardware design described by the original HDL code. This download can be either done through a serial connection, a JTAG cable or programmed into a ROM and loaded into the FPGA every time power is applied.

Appendix C

FPGA Implementation

The proposed prototype architectures described in Chapters 3, 4 and 5 have been described in VHDL [vhd93, Bha99, Per01], synthesized under the FPGA Advantage 5.0V [fad] and the XST (Xilinx Synthesis Tool) [xil] and then implemented on the Virtex FPGA device. The synthesis and implementation results for a few individual blocks are provided in this appendix.

C.1 Synthesis Results for Major Functional Blocks

The individual blocks have been simulated on Modelsim 5.5b and then synthesized at gate level on XCV100-6PQ240.

VHDL simulator:	ModelSim 5.5b	
Synthesis Tools:	(i) Xilinx Synthesis Tool (XST) (ii) FPGA Express (iii) Leonardo Spectrum	
FPGA device targeted:		
Family	**Device**	**Grade**
Virtex:	XCV100HQ240	-6

Table C.1: Software Tools.

FIFO Circuit

The gate level circuit diagram of the synchronous FIFO (vide Figure 3.16 of Chapter 3) is shown in Figure C.1. The performance of the 255×8 FIFO is limited to 139.76MHz. It utilizes 1% of CLB slices.

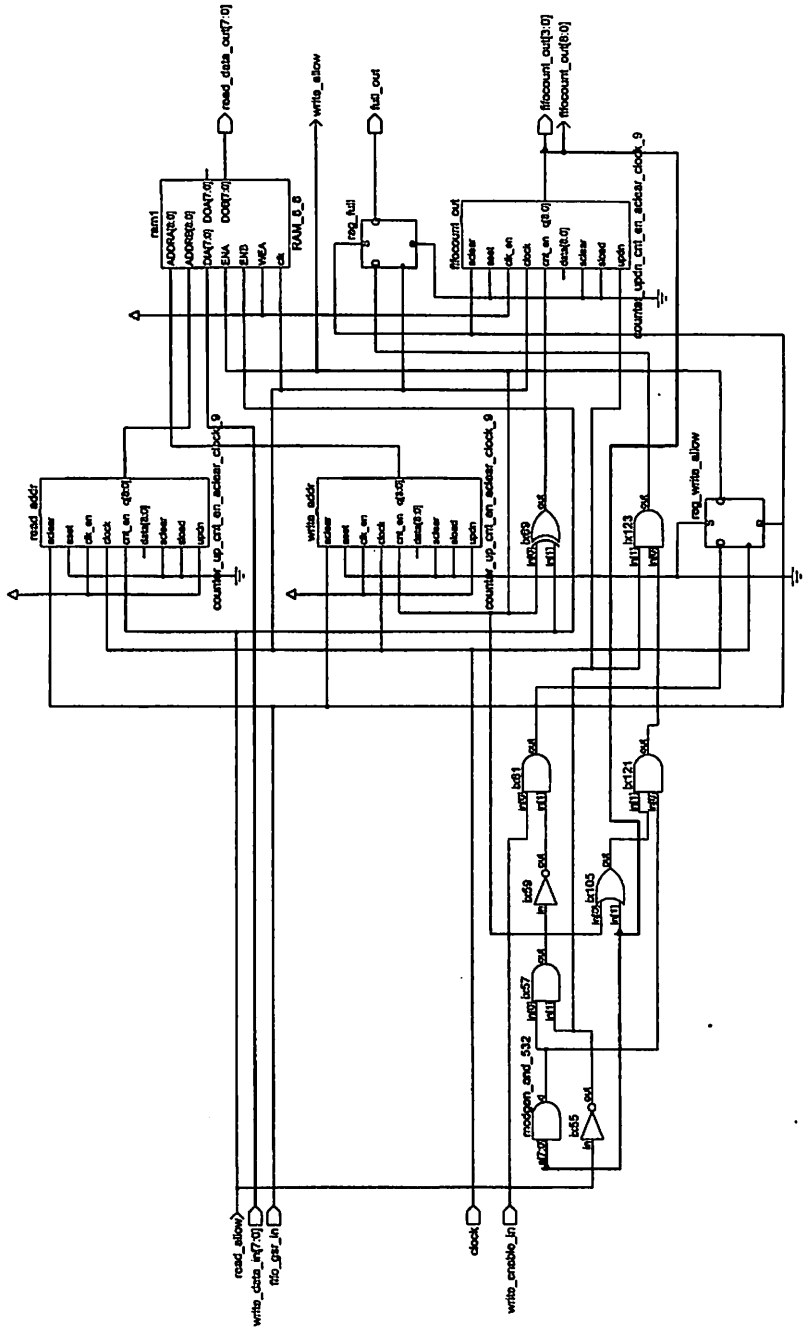


Figure C.1: The synthesized circuit for FIFO implementation.

Neighborhood Address Generation Block (NAG)

The synthesized circuit diagram of the NAG block (vide Figure 3.17 of Chapter 3) is shown in Figure C.2. The neighborhood addresses (N_ADDR_1 – N_ADDR_7) are generated in parallel, involving addition/subtraction of the image width W and the B_ADDR. The performance of the NAG block is limited to 56.92MHz. It utilizes 6% of CLB slices.

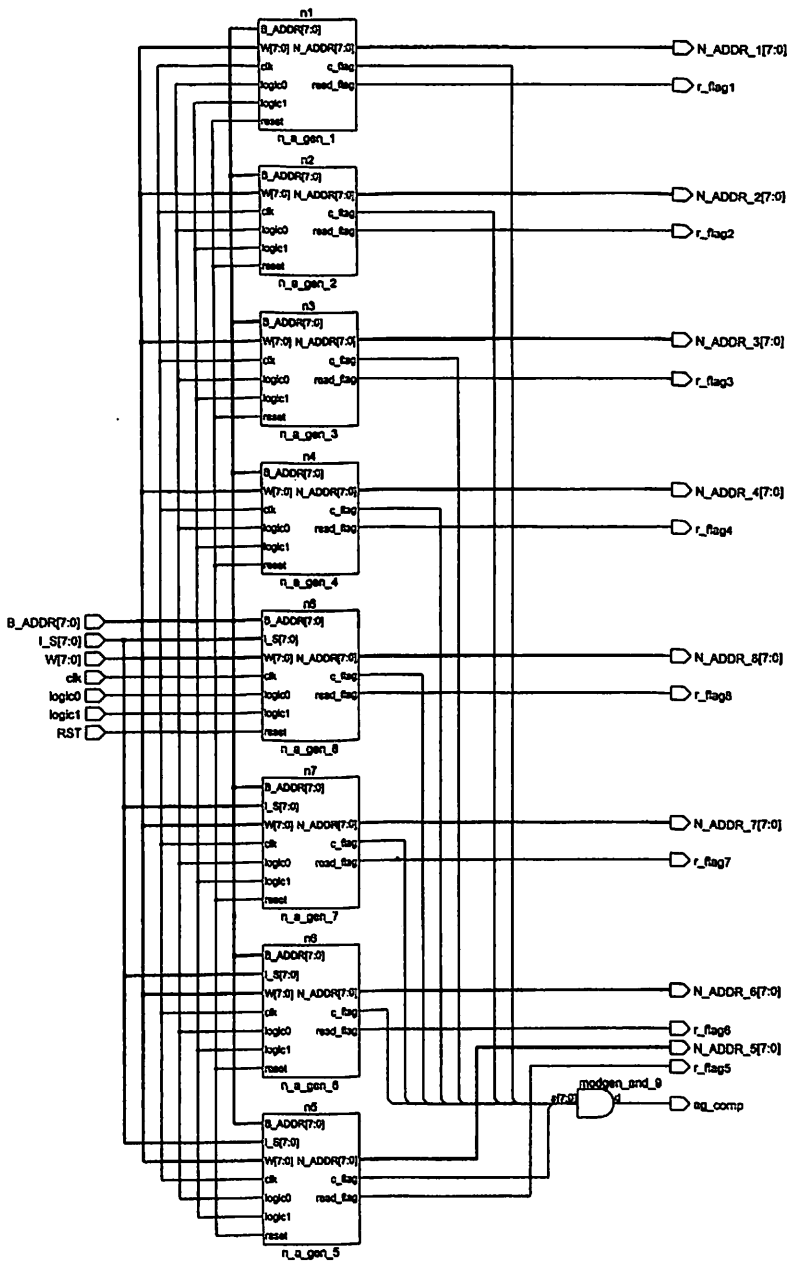


Figure C.2: The synthesized circuit diagram of the NAG block.

Plateau Detection (PD) Block

The synthesized circuit diagram of the PD block (vide Figure 3.18 of Chapter 3) is shown in Figure C.3. The plateau detection is performed by comparing the center pixel value B_DATA with each of its neighboring pixel values (N_data_1 – N_data_8) in parallel. The performance of the PD block is limited to 81.24MHz. It utilizes 1% of CLB slices.

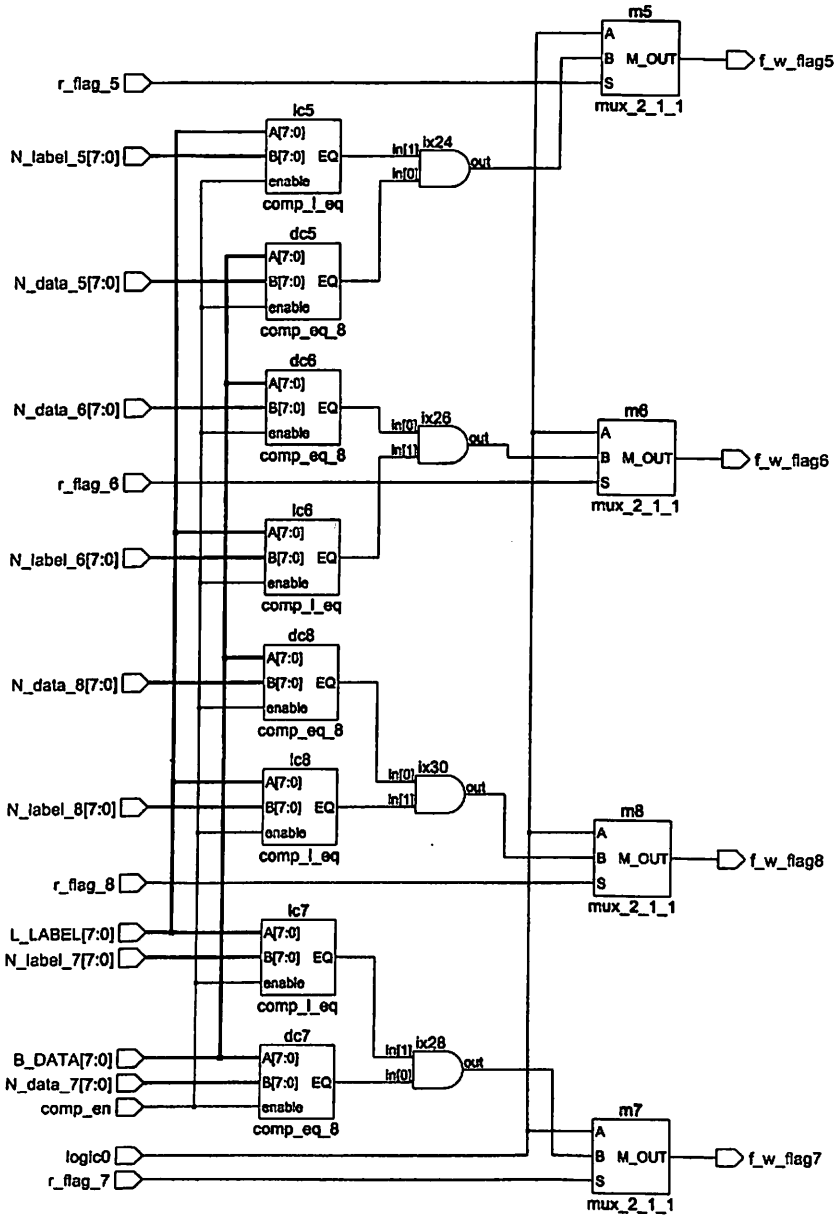


Figure C.3: The synthesized circuit diagram of the PD block.

Plateau Analysis (PA) Block

The synthesized circuit diagram of the PA block (vide Figure 3.19 of Chapter 3) is shown in Figure C.4. It is used to detect whether the center pixel b_data belongs to the non-minima plateau. The performance of the PA block is limited to 65.38MHz. It utilizes 1% of CLB slices. The internal structure of a p_a

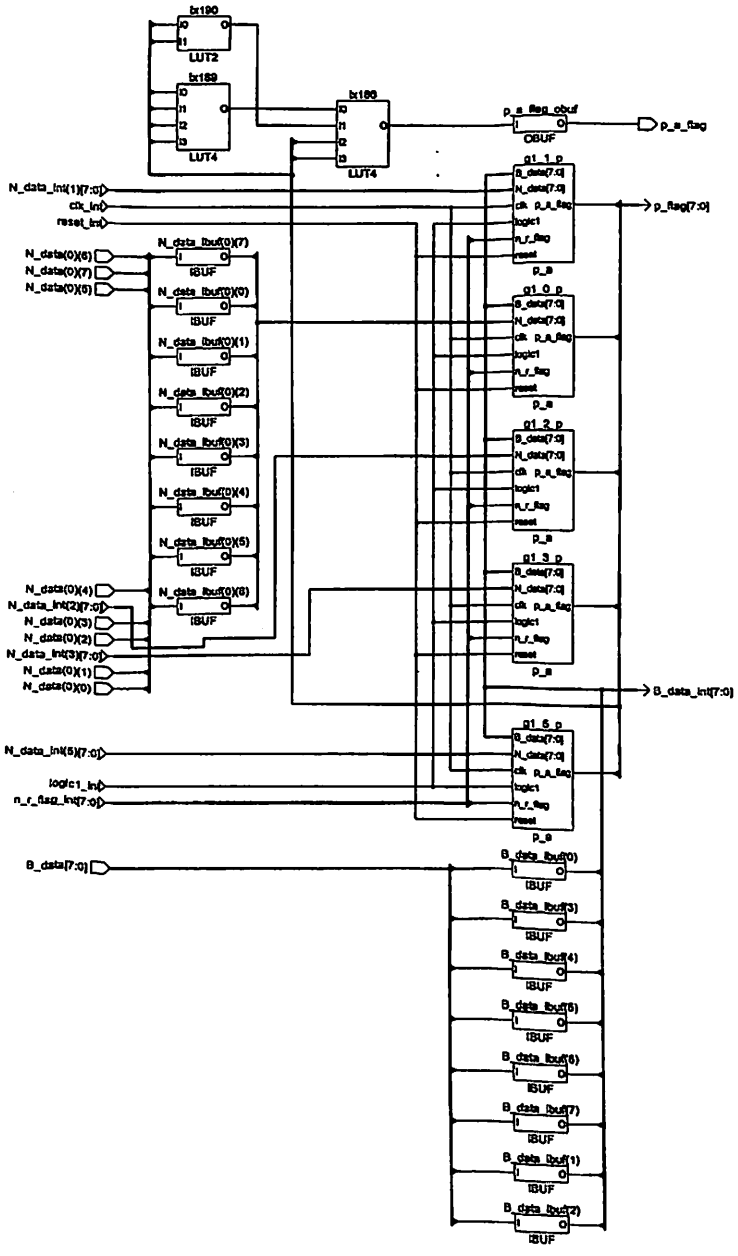


Figure C.4: The synthesized circuit diagram of the PA block.

block is provided next.

p_a block

Figure C.5 depicts the synthesized circuit diagram of the p_a block, which compares the center pixel b_data with the neighboring pixel n_data. The p_flag signal indicates that the center pixel b_data belongs to the non-minima plateau.

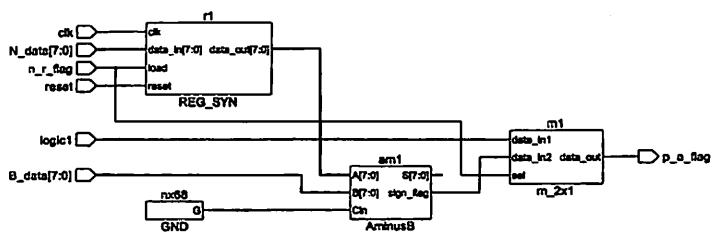


Figure C.5: The synthesized circuit diagram of the p_a block.

Memory Address Generation Counter Block

The synthesized circuit diagram of the 9-bit memory address counter is shown in Figure C.6, which is used to perform the raster scan in the image.

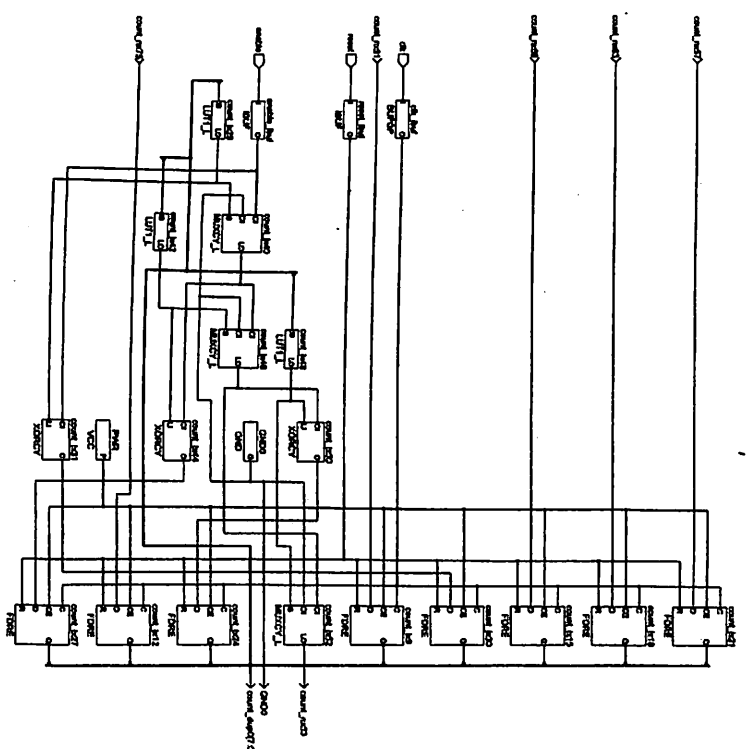


Figure C.6: The synthesized circuit diagram of the Memory Address Generation Counter Block.

The PE block used in Conditional Neighborhood Comparison Block (vide Figure 5.23 of Chapter 5) may be given as follows.

Processing Element (PE)

The synthesized circuit diagram of the PE block is shown in Figure C.7. The labeled B_N module block is used to detect the validity of the deciding neighboring pixel (n_data). The validity signal f_w_flag is generated when the neighboring pixel (n_data) has its intensity value greater than that of the center pixel (b_data), the distance (n_dist) equal to one more than the center pixel distance (b_dist), and the neighboring pixel label (n_label) happens to be NARM. The seven blocks n_n_n_1, n_n_n_2, . . ., n_n_n_7 are used to decide the label of the neighboring pixel (n_data) based on four different conditions (vide Figure 5.8) of Chapter 5.

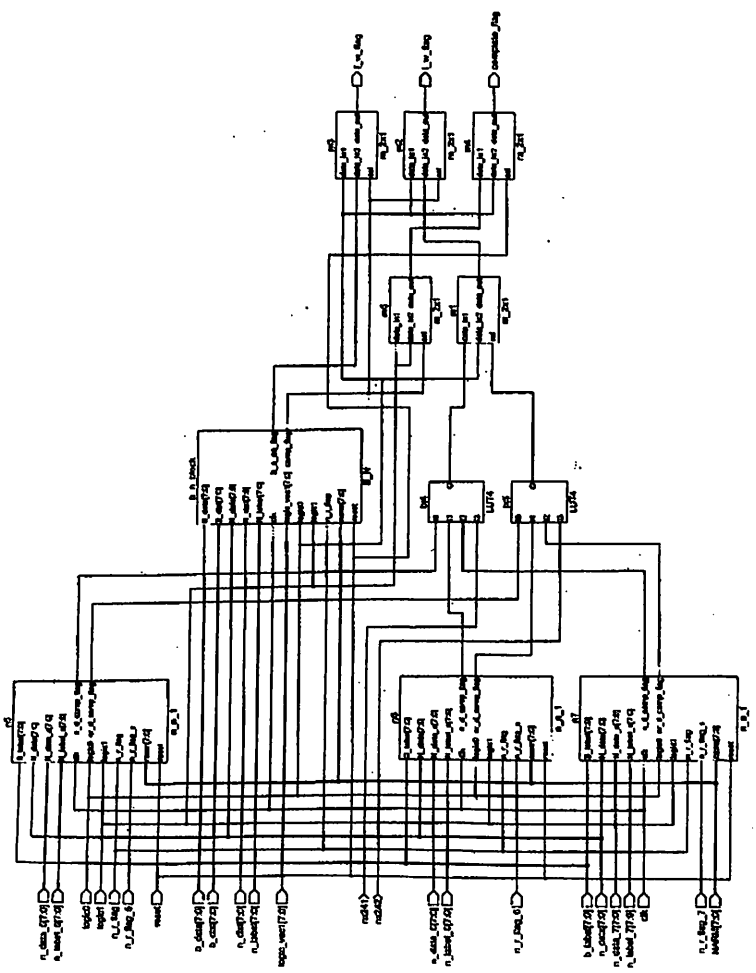


Figure C.7: The synthesized circuit diagram of the PE block

C.2 Floor Planner and FPGA Editor Results after Place and Route

The proposed prototype architecture for hillclimbing technique described in Chapter 4 has been implemented on the FPGA chip XCV100PQ240. Figure C.8 shows the Floor Planner results while placing. Figure C.9 shows the FPGA Editor results after routing.

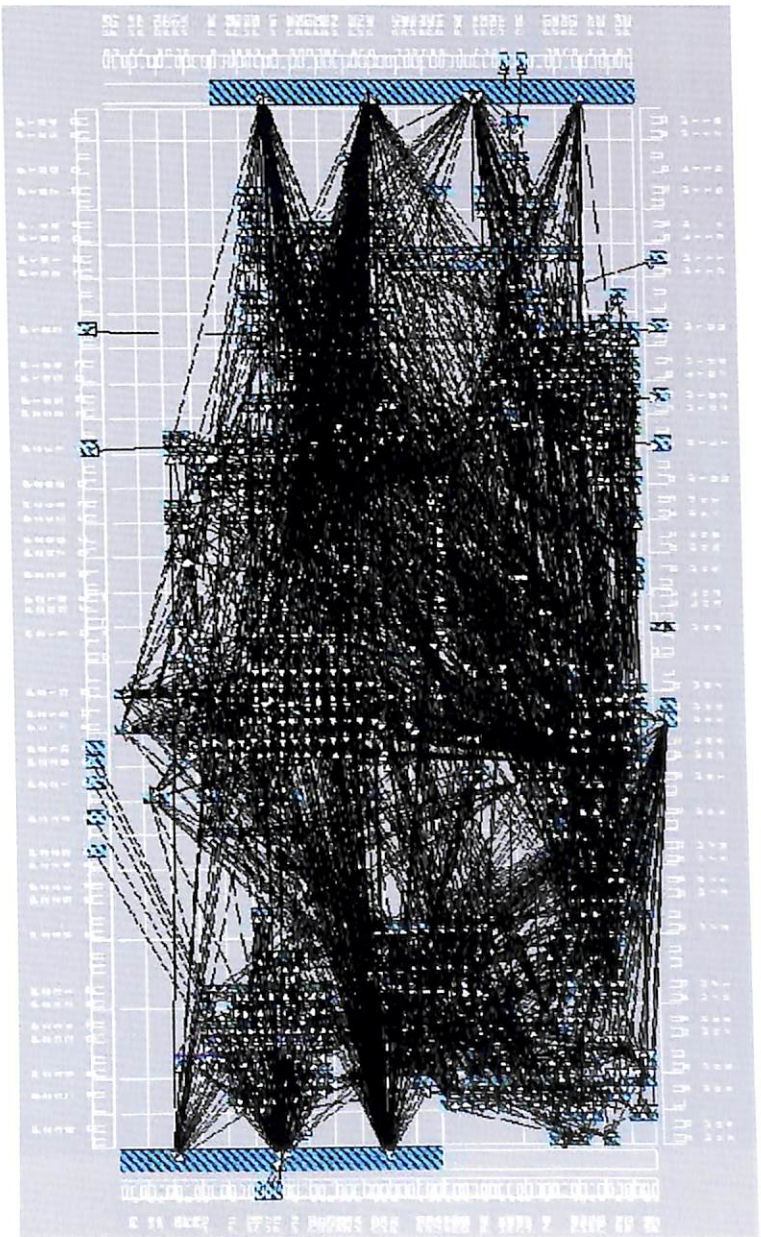


Figure C.8: Placed Design (Floor Planner).

Bibliography

[ADR95] H. Ancin, T. E. Dufresne, G. M. Ridder, J. N. Turner, and B. Roysam. An improved watershed algorithm for counting objects in noisy, anistropic 3-D biological images. In *Proceedings of IEEE International Conference on Image Processing*, volume III, pages 172–175. Washington, D. C., October 1995.

[alt] User Software manuals. Technical report. http://www.altera.com/support/spt-index.html.

[ARH80] N. Ahuja, A. Rosenfeld, and R. M. Haralick. Neighbour gray levels as features in pixel classification. *Pattern Recognition*, 12:251–260, 1980.

[BA95] S. M. Bhandarkar and H. R. Arabnia. The hough transform on a reconfigurable multi-ring network. *Journal of Parallel and Distributed Computing*, 24(1):107–114, January 1995.

[BA97] S. M. Bhandarkar and H. R. Arabnia. Parallel computer vision on a reconfigurable multiprocessor network. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):292–309, March 1997.

[BBM97] A. Bieniek, H. Burkhardt, H. Marschner, M. Nolle, and G. Schreiber. A parellel watershed algorithm. In *Proc. 10th Scandinavian Conference on Image Analysis (SCIA'97)*, pages 237–244, Lappeenranta, Finland, 1997.

[BBY92] B. Beucher, M. Bilodeau, and X. Yu. Road tracking, lane segmentation and obstacle recognition by mathematical morphology. In *Proceedings of Intelligent Vehicles'92 Symposium*, Detroit, USA, 1992.

[Beu90a] S. Beucher. *Segmentation d'images et morphologie mathématique*. PhD thesis, School of Mines, Paris, June 1990.

[Beu90b] S. Beucher. The watershed tools in mathematical morphology. In *SPIE Congress*, San Diego, 1990.

[BF82] B. Banu and O. D. Faugeras. Segmentation of images having unimodel distributions *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 4(4):408–419, 1982.

[BG89] J. M. Beaulieu and M. Goldberg. Hierarchy in picture segmentation: a stepwise optimization approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):150–163, 1989.

[Bha99] J. Bhaskar. *VHDL primer*. Prentice Hall, third edition edition, 1999.

[BL79] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *International Workshop on image processing: Real-time Edge and Motion/estimation*, pages 2.1–2.12, Rennes, France, September 1979.

[BM93] S. Beucher and F. Meyer. The morphological approach to segmentation: The watershed transformation. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 433–481. Marcel Dekker Inc.,New York, 1993.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, November 1986.

[CCNC97] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata Theory and Applications*, volume 1, chapter 10, pages 274–313. IEEE Computer Society Press, Los Alamitos, California, 1997.

[CIA04] C. Rambabu, I. Chakrabarti, and A. Mahanta. A novel flooding-based watershed algorithm and its prototype hardware architecture. *To appear in IEE Proceedings - Vision, Image and Signal Processing*, 2004.

[CKP98] B. Chanda, M. K. Kundu, and Y. V. Padmaja. A multi-scale morphological edge detector. *Pattern Recognition*, 31(10):1469–1478, January 1998.

[CL94] Y. L. Chang and X. Li. Adaptive image region growing. *IEEE Transactions on Image Processing*, 3(6):868–873, 1994.

[CSS98] M. Cheriet, J. N. Said, and C. Y. Suen. A recursive thresholding technique for image segmentation. *IEEE Transaction on Image Processing*, 7(6):918–920, June 1998.

[Dav75] L. S. Davis. A survey of edge detection techniques. *Computer Vision, Graphics, Image Processing*, 4:248–270, 1975.

[DE87] H. Derin and H. Elliott. Modeling and segmentation of noisy and textured images using gibbs random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:39–55, 1987.

[Deh00] A. Dehon. The density advantage of configurable computing. *Computer*, 33(4):41–49, April 2000.

[DJ89] R. Dubes and A. Jain. Random field models in image analysis. *Journal of Applied Statistics*, 16:131–164, 1989.

[DJNC90] R. C. Dubes, A. K. Jain, S. G. Nadabar, and C. C. Chen. MRF model-based algorithms for image segmentation. In *Proceedings of 10th International Conference on Pattern Recognition*, volume 1, pages 808–814. Atlantic City, NJ, USA, June 1990.

[EG03] M. Edwads and P. Green. Run-time support for dynamically reconfigurable computing systems. *Journal of System Architecture*, 49:267–281, 2003.

[fad] User Software manuals. Technical report. http://www.mentor.com/fpga-advantage/.

[GD88] Charles R. Giadina and Edward R. Dougherty. *Morphological Methods in Image and Signal Processing*. Prentice-Hall, Inc., 1988.

[Gee95] L. A. Gee. Segmentation of range images using morphological operators: review and examples. In *SPIE Conference on Intelligent Robots and Computer Vision XIV*, volume 2588, pages 734–746. Philadelphia, PA, Oct. 1995.

[Gro97] MPEG-4 Video Group. Mpeg-4 video verification model version 7.0. In *ISO/IEC JTC1/SC29/WG11,MPEG97/N1642*. Bristol, England, Apr. 1997.

[GW01] Rafael C. Gonzalez and Richard E. Woods. *Edge Detection*, chapter 7, pages 413–443. Pearson Education Pte. Ltd., INC., seventh edition, 2001.

[HD95] Q. Huang and B. Dom. Quantitative methods for evaluating image segmentation. In *IEEE International conference on Image Processing*, volume 3, pages 53–56, Washington D. C., October 1995.

[HS85] R. Haralick and L. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, Image Processing*, 29(1):100–132, January 1985.

[HSZ87] R. Haralick, S. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:532–550, 1987.

[KAD03] L. Kessel, N. Abel, and D. Demigny. Real-time image processing with dynamically reconfigurable architecture. *Real-Time Imaging*, 9:297–313, 2003.

[KK03] J. B. Kim and H. J. Kim. Multiresolution-based watersheds for efficient image segmentation. *Pattern Recognition Letters*, 24:473–488, 2003.

[Kur91] T. Kuria. An efficient agglomerative clustering algorithm using a heap. *Pattern Recognition*, 24(3):205–209, 1991.

[LM84] C. Lantuéjoul and F. Maisonneuve. Geodesic methods in quantitative image analysis. *Pattern Recognition*, 17:117–187, 1984.

[MB90] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–45, 1990.

[MCG95] Alina N. Moga, Bogdan Cramariue, and Moncef Gabbouj. An efficient watershed segmentation algorithm suitable for parallel implementation. In *Procedings of IEEE International Conference on Image Processing*, volume 2, pages 101–104, October 1995.

[Mey94] F. Meyer. Topographic distance and watershed lines. *Signal Processing*, 38(1):113–125, July 1994.

[MG97] Alina N. Moga and Moncef Gabbouj. Parallel image component labeling with watershed transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):441–450, May 1997.

[MH88] K. Mardia and T. Hainsworth. A spatial thresholding method for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:919–927, 1988.

[Mog97] Alina N. Moga. *Parallel watershed algorithms for image segmentation*. PhD thesis, Tampere University of Technology, Tampere, Finland, February 1997.

[MPL00] M. R. Rezaee, P. M. J. van der Zwet, B. P. E. Leieveldt, R. J. van der Geest, and J. H. C. Reiber. A multiresolution image segmentation technique based on pyramidal segmentation and fuzzy clustering. *IEEE Transaction on Image Processing*, 9(7):1238–1248, 2000.

[MVG95] AliN. Moga, T. Viero, M. Gabbouj, M. Nolle, G. Schreiber, and H. Burkhardt. Parallel watershed algorithm based on sequential scanning. In I. Pitas, editor, *Proc. IEEE Workshop on Nonlinear Signal and Image Processing*, pages 991–994, Neos Marmaras, Halkidiki, Greece, June 1995.

[NC03] L. Najman and M. Couprie. Watershed algorithms and contrast preservation. In *Discrete Geomentry for Computer Imagery*, volume 2886 of *Lecture Notes in Computer Science*, pages 62–71. Springer-Verlag Heidelberg, November 2003.

[OM97] J. R. Ohm and P. Ma. Feature-based cluster segmentation of image sequences. In *Proc. IEEE international Conference on Image Processing*, pages 178–181, 1997.

[Ots78] N. Otsu. A threshold selection method from grey level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 8:62–66, 1978.

[Per80] W. A. Perkins. Area segmentation of images using edge points. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 2(1):8–15, 1980.

[Per01] Douglas L. Perry. *VHDL*. Tata McGraw-Hill Publishing Company Limited, third edition edition, 2001.

[PF99] J. Pauwels and G. Frederix. Finding salient regions in images. *Computer Vision and Image Understanding*, 75:73–85, 1999.

[PP93] Nikhil R. Pal and Sankar K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[Pra80] J. M. Prager. Extracting and labeling boundary segments in natural scenes. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 2(1):16–27, 1980.

[Pra03a] William K. Pratt. *Edge Detection*, chapter 15, pages 443–508. John Willey and sons, INC., third edition, 2003.

[Pra03b] William K. Pratt. *Image segmentation*, chapter 17, pages 551–588. John Willey and sons, INC., third edition, 2003.

[RC03] C. Rambabu and I. Chakrabarti. An improved immersion-based watershed transform method. In *Proceedings of the 5th International Conference on Advances in Pattern Recognition (ICAPR 2003)*, pages 192–195, Kolkata, India, December 2003.

[RCG03] C. Rambabu, I. Chakrabarti, and D. Ghosh. An efficient watershed transform computation method. In *Fourth IEEE Pacific-Rim Conference on Multimedia*, Singapore, December 2003.

[RCM02] C. Rambabu, I. Chakrabarti, and A. Mahanta. An efficient architecture for an improved watershed algorithm and its FPGA implementation. In *IEEE International Conference on Field Programmable Technology*, pages 370–373. Hong Kong, December 2002.

[RCM03] C. Rambabu, I. Chakrabarti, and A. Mahanta. An improved flooding based watershed algorithm. In *Ninth National Conference on Communications (NCC'2003)*, pages 90–94. IIT Madras, India, February 2003.

[RDKJ02] Cecila Di Ruberto, Andrew Dempster, Shahid Khan, and Bill Jarra. Analysis of infected blood cell images using morphological operators. *Image and Vision Computing*, 20(2):133–146, February 2002.

[RM01] Jos B. T. M. Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, 2001.

[RRC03] C. Rambabu, T. S. Rathore, and I. Chakrabarti. A new watershed algorithm based on hillclimbing technique for image segmentation. In *Proceedings of IEEE TENCON 2003*, pages 1404–1408. Bangalore, India, October 2003.

[SAB99] B. Sankur, A. T. Abak, and U. Baris. Assessment of thresholding algorithms for document processing. In *Proceedings of IEEE International Conference on Image Processing*, volume 1, pages 580–584, Kobe, Japan, October 1999.

[Sal94] P. Salembier. Morphological multiscale segmentation for image coding. *Signal Processing*, 38(3):359–386, August 1994.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., 1982.

[SS89] C. Srinivas and M. D. Srinath. Compound gaussian markov random field model for image segmentation. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1586–1589. Glasgow,UK, May 1989.

[Ste86] S. Sternberg. Gray scale morphology. *Computer Vision, Graphics, and Image processing*, 35:333–355, 1986.

[SYL03] Shao-Yi Chien, Yu-Wen Huang, and Liang-Gee Chen. Predictive watershed: a fast watershed algorithm for video segmentation. *IEEE Transaction on Circuits and Systems for video technology*, 13(5):453–461, May 2003.

[Tri94] Stephen M. Trimberger. *Field Programable Gate Array Technology*, chapter 1-4, pages 10–134. Kluwer Academic Publishers, Boston, 1994.

[vhd93] *IEEE Standard VHDL Language Reference Manual*. The Institute of Electrical and Electronics Engineers, 1993.

[Vie96] T. Viero. *Algorithms for image sequence filtering, coding and image segmentation*. PhD thesis, Tampere University of Technology, Tampere, Finland, January 1996.

[Vin90] Luc Vincent. *Morphological algorithms based on Queues and loops, with extension to graphs*. PhD thesis, Cahiers due Centre de morphologie Mathématique, School of Mines, Paris, May 1990.

[Vin93] Luc Vincent. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201, April 1993.

[VS91] L. Vincent and P. Soillé. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991.

[WA03] M. A. Wani and H. R. Arabnia. Parallel edge-region-based segmentation algorithm targeted at reconfigurable multiring network. *The Journal of Supercomputing*, 25(1):43–62, May 2003.

[Wan97] D. Wang. A multiscale gradient algorithm for image segmentation using watersheds. *Pattern Recognition*, 30(12):2043–2052, January 1997.

[Wan98] D. Wang. Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Tansactions on Circuits and Systems for Video Technology*, 8(5):539–546, September 1998.

[Wes78] J. S. Weska. A survey of threshold selection techniques. *Computer graphics and image processing*, 7(2):259–265, April 1978.

[WH98] M. J. Wirthlin and B. L. Hutchings. Improving functional density using run-time circuit reconfiguration. *IEEE Transaction on VLSI Systems*, 6(2):247–256, June 1998.

[WHOF96] S. Wegner, T. Harms, H. Oswald, and E. Fleck. Medical image segmentation using the watershed transformation on graphs. In *Proceedings of International Conference on Image Processing*, volume 3, pages 37–40. Lausanne, Switzerland, Sept. 1996.

[WLGW01] J. Z. Wang, J. Li, R. M. Gray, and G. Wiederhold. Unsupervised multiresolution segmentation for images with low depth of field. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23(1):85–90, 2001.

[WMTM95] V. Warscotte, B. Macq, J.-P. Thiran, and C. Michel. Accurate segmentation of 3-D magnetic resonance images of the head using a directional watershed transform. In *IEEE*

17th Annual Conference on Engineering in Medicine and Biology Society, volume 3, pages 491–492. Montreal, Que, Canada, September 1995.

[WNR74] J. S. Weska, R. N. Nagel, and A. Rosenfield. A threshold selection technique. *IEEE Transactions on Computers*, C-23:1322–1326, December 1974.

[Wu93] X. Wu. Adaptive split-and-merge segmentation based on piecewise least-square approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:808–815, August 1993.

[x1] Xilinx website. Technical report. http://www.xilinx.com.

[xil] User Software manuals. Technical report. http://Support.xilinx.com/support/sw_manuals/.

Related Publications by the Author

Journal Publication:

1. C. Rambabu, I. Chakrabarti and A. Mahanta, "Novel flooding based watershed algorithm and its prototype hardware architecture," *to appear in IEE Proceedings - Vision, Image and Signal Processing*, 2004.

Communicated to Journals:

1. C. Rambabu, T. S. Rathore and I. Chakrabarti, "A fast hillclimbing-based watershed algorithm for image segmentation, **IETE Journal**.
2. C. Rambabu and I. Chakrabarti, "An efficient immersion-based watershed transform method and its prototype architecture," **Journal of System Architecture**.

Conference Publications:

1. C. Rambabu, I. Chakrabarti and A. Mahanta, "An efficient architecture for an improved watershed algorithm and its FPGA implementation," in *Proceedings of IEEE International conference on Field-Programmable Technology*, Hong Kong, December 2002, pp. 370-373.
2. C. Rambabu, I. Chakrabarti and A. Mahanta, "An improved flooding based watershed algorithm," in *Proceedings of Ninth National Conference on Communications(NCC)*, IIT Chennai, February 2003, pp. 90-94.
3. C. Rambabu, T. S. Rathore and I. Chakrabarti, "A new watershed algorithm based on Hillclimbing technique for image segmentation," in *Proceedings of IEEE TENCON-2003*, Bangalore, India, October 2003, pp. 1404-1408.
4. C. Rambabu, I. Chakrabarti and D. Ghosh, " An improved immersion-based watershed transform method," in *Proceedings of the 5th International Conference on Advances in Pattern Recognition (ICAPR 2003)*, Kolkata, India, December 2003, pp. 192-195.
5. C. Rambabu, I. Chakrabarti and D. Ghosh, "An efficient watershed transform computation method, in *Proceedings of ICICS & PCM - 2003*, Singapore, December 2003.

Bio-data of the Author

Mr. C. Rambabu received the Bachelor of Engineering degree in Electronics and Communication Engineering in 1995 from Andhra University, Visakhapatnam. He acquired the Master of Technology in Electronics and Electrical Communication Engg. from IIT Kharagpur in 1998. He next joined the PhD program in Electronics and Communication Engg. in IIT Guwahati. His research interests lie in Image Processing, Pattern Recognition, Computer Vision and VLSI Digital Signal Processing.