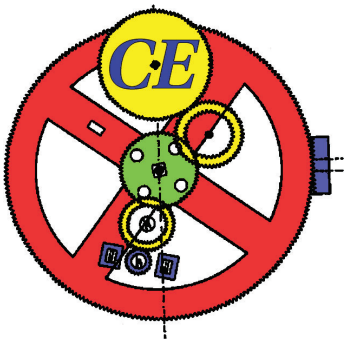

Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2009

MSc THESIS

Implementing Texture Feature Extraction Algorithms on FPGA

Mahshid Roumi

CE-MS-2009-25

Abstract

Feature extraction is a key function in various image processing applications. A feature is an image characteristic that can capture certain visual property of the image. Texture is an important feature of many image types, which is the pattern of information or arrangement of the structure found in a picture. Texture features are used in different applications such as image processing, remote sensing and content-based image retrieval. These features can be extracted in several ways. The most common way is using a Gray Level Co-occurrence Matrix (GLCM). GLCM contains the second-order statistical information of neighboring pixels of an image. Textural properties can be calculated from GLCM to understand the details about the image content. However, the calculation of GLCM is very computationally intensive. In this thesis, an FPGA accelerator for fast calculation of GLCM is designed and implemented. We propose an FPGA-based architecture for parallel computation of symmetric co-occurrence matrices. Experimental results show that our approach improves 2x up to 4x the processing time for simultaneous computation of sixteen co-occurrence matrices.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Implementing Texture Feature Extraction Algorithms on FPGA

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Mahshid Roumi
born in Tehran, Iran

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Implementing Texture Feature Extraction Algorithms on FPGA

by Mahshid Roumi

Abstract

Feature extraction is a key function in various image processing applications. A feature is an image characteristic that can capture certain visual property of the image. Texture is an important feature of many image types, which is the pattern of information or arrangement of the structure found in a picture. Texture features are used in different applications such as image processing, remote sensing and content-based image retrieval. These features can be extracted in several ways. The most common way is using a Gray Level Co-occurrence Matrix (GLCM). GLCM contains the second-order statistical information of neighboring pixels of an image. Textural properties can be calculated from GLCM to understand the details about the image content. However, the calculation of GLCM is very computationally intensive. In this thesis, an FPGA accelerator for fast calculation of GLCM is designed and implemented. We propose an FPGA-based architecture for parallel computation of symmetric co-occurrence matrices. Experimental results show that our approach improves 2x up to 4x the processing time for simultaneous computation of sixteen co-occurrence matrices.

Laboratory : Computer Engineering
Codenumber : CE-MS-2009-25

Committee Members :

Advisor:	Dr. Ir. Georgi Gaydadjiev , CE, TU Delft
Advisor:	Dr. Ir. A. Shahbahrami , CE, TU Delft
Chairperson:	Dr. Koen Bertels , CE, TU Delft
Member:	Dr. Ir. Zaid Al-Ars , CE, TU Delft
Member:	Dr. Ir. Rene van Leuken, ENS, TU Delft

To My Parents and my Brother for their unconditional love and endless support

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I am grateful to my parents, and my brother for their unlimited and unconditional love, encouragement and support in my whole life.
My special thanks goes to my supervisor Dr. Georgi Gaydadjiev for his patience, time, and help.
Special thanks also to Dr. Asad Shahbahami for his advice, support, and encouragement during my MSc project.
My special thanks goes to all my friends warmly for their support in all aspects during my time in Delft. I cannot forget my great time in Delft with them.
I would also like to thank all CE members for their support, and I am thankful to you too.

Mahshid Roumi
Delft
August 2009

1 Introduction

A key function in different image applications is feature extraction. A feature is a characteristic that can capture a certain visual property of an image either globally for the whole image, or locally for objects or regions. Different features such as color, shape, and texture can be extricated from an image. Texture is the variation of data at different scales. A number of methods have been developed for texture feature extraction. They can be extracted from co-occurrence matrices and wavelet transform coefficients. There after, they are stored as feature vectors.

In this thesis we focus on calculating the co-occurrence matrices. As calculating this matrix and features are time consuming, we design a dedicated hardware on FPGA for this purpose.

This chapter is organized as follows. Section 1.1 presents the motivation for this work. Our project goals are identified in Section 1.2. Section 1.3 describes the thesis organization.

1.1 Motivation

There are different algorithms to extract texture features such as structural, statistical, and transform domain. The structural approaches provide symbolic description for an image. The statistical approaches provide texture features by distribution and relationships between the gray levels of an image. In addition, texture features can be extracted using transform coefficients such as wavelet transform coefficients.

1.2 Project Goals

- To investigate different approaches to extract texture features extraction,
- To evaluate the co-occurrence matrix as a suitable technique for texture feature extraction. In addition, to investigate the computation of this matrix,
- To design dedicated hardware to speedup the calculation of the co-occurrence matrix and extracting features from it.
- To use three platforms FPGA to compare their results and performances.

1.3 Thesis Organization

The thesis is organized as follows. In Chapter 2, we discuss the basic information about digital image processing, texture features and different techniques that can be used for

texture feature extraction. We present related work in Chapter 3 followed by hardware implementation in Chapter 4. We present the summary and the future work in Chapter 5. Last chapter is bibliography.

Digital Images and Texture Features

2

This chapter contains an overview of a digital image, and texture. Section 2.1 describes the digital image. Image analysis is defines in section Section 2.2. The following sections will provide a discussion about texture and different methods to extract texture features.

2.1 Digital Image

An image is defined as a two dimensional function, $f(x, y)$, where x and y are spatial coordinates, and $f(x, y)$ is a set of G grey-tones. When x, y and the grey-tones of f have discrete quantities, the image is called a digital image [14].

The function $f(x, y)$ is [14]:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N_y-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N_y-1) \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ f(N_x-1,0) & f(N_x-1,1) & \cdots & f(N_x-1,N_y-1) \end{bmatrix} \tag{2.1}$$

A digital image has a finite number of elements, each of these elements has a particular value and location. These elements are called pixel or image elements [14].

Digital image has a different types, some of them are [45]:

- Binary image:

  This is the simplest type of image with two gray-values, 0 and 1 or black and white. Each pixel is represented by a single bit. These types of images are useful in computer vision applications where only information about images or outlines are required. It can be created from gray-scale image that uses 0 for pixels with gray levels below the threshold and 1 for other pixel but this way of creation is not useful because most of the information is lost and the image result are smaller.

- Gray-scale image:

  These images contain the brightness information. The number of bits that are used to represent each pixel, are related to the number of different brightness levels available. The typical image contains 8 bits per pixel so there are 256 different possible gray-tones (N_g) or intensity values from 0 to 255.

- Color image:

  Normally, images are represented as RGB (Red, Green, Blue) models, and each pixel has 24 bits. The brightness information and color information are coupled and represented in many applications. These two information are septated by transferring RGB information to the mathematical.

Digital image can be presented in different ways. The first way is plot image as a surface with three dimensions (x, y, z), in which x, y are spatial coordinates, and z is the value of the $f(x, y)$. In the second way, the image is shown as a visual intensity array. There are just three intensity values, 0 (black), 0.5 (gray), 1 (white). In the third method, the image is represented as a 2D array. This is useful when the size of the image is big and one part of it needs to be analyzed.

2.2 Image Analysis

Image analysis considers the image for a specific application and it is useful in many applications such as computer vision, medical imaging, and pattern recognition. It extracts the useful information from the image. The image analysis involves image segmentation, image transformation, pattern classification, and feature extraction [45]. Some of these concepts are explained in the remainder of this chapter.

- Image segmentation: Image segmentation divides the input image into multiple segments or regions, which show objects or meaningful parts of objects. It segments image into homogenous regions thus making it easier to analyze images.

- Image transformation: Image transform is used to find the spatial frequency information that can be used in the feature extraction step.

- Feature extraction: Large amounts of data are used to represent an image, so the analysis of an image needs a large amount of memory and thus takes more time. In order to reduce the amount of data, an image is represented using a set of features. Feature extraction is a primitive type of pattern recognition and it is very important for pattern recognition. This step extract some features such as color, shape, and texture. Features contain the relevant information of an image and will be used in image processing (e.g. searching, retrieval, storing). It decomposes into two parts, feature construction and feature selection.

  Features are divided into different classes based on the kind of properties they describe. Some important features are as follows.

  - Color:

    From a mathematical viewpoint the color signal is an extension from scalar-signals to vector-signals. The color can be represented by an average color (three scalars) or a color histogram (three functions). Color features can be derived from a histogram of the image. The multi-dimensional histogram is used for the multi-color image [32]. From a brightness histogram, the color features are derived as RGB components. Color features are useful for biomedical image processing, such as for cell classification, and cancer cell detection [32]. The weakness is that the color histogram of two different things with the same color, are equal.

  - Texture:

There is no unique definition of texture, texture is a characteristic of an image that can provide a higher-order description of the image and includes information about the spatial distribution of tonal variations or gray tones [17]. An image can have one or more textures. These features are useful in many applications such as in medical imaging [42], remote sensing [39], and content based image retrieval.

– Structure:
This feature describe the structure of the image or the locations of objects in the image in contrast to each other. For example, an image includes different objects such as tree, bird, cat etc.

– Shape
From the human point of view shape is a high-level concept but from the mathematical view shape is a low-level element. In pattern recognition, shape is described as a function of position and direction simply as a connected curve within a 2D field [7]. Shape features can be used for medical applications for example for cervical cell classification or for content-based image retrieval systems where color features are not useful [8].

2.3 Texture

2.3.1 Introduction of Texture

There are different definitions of texture, some of them are as follows.

- Texture plays an important role in human vision and in image classification. Pictures of flowers, walls, water or patterns on a fabric or single objects are distinguished according to their texture. The observation of texture depends on certain conditions such as light, angle, distance, or other environmental effects [30].
- Texture is the pattern of information or arrangement of the structure found in a picture [25].
- Texture uses features in the analysis and interpretation of images. It can be characterized as a set of local statistical properties of the pixel gray level intensity [40].
- Texture is one of the important characteristics in identifying objects or regions of interest in an image [17].
- There are two types of texture based on spatial frequency, namely, fine and coarse. Fine textures have high spatial frequencies or a high number of edges per unit area. Coarse textures have low spatial frequencies or a small number of edges per unit area [16].

Figure 2.1 depicts some examples of different texture features.

Textures are classified into two classes, namely, touch and visual. Touch textures relate to the touchable feel of a surface and range from the smoothest (little difference between high and low points) to the roughest (large difference between high and low points). Visual textures relate to human image observation.

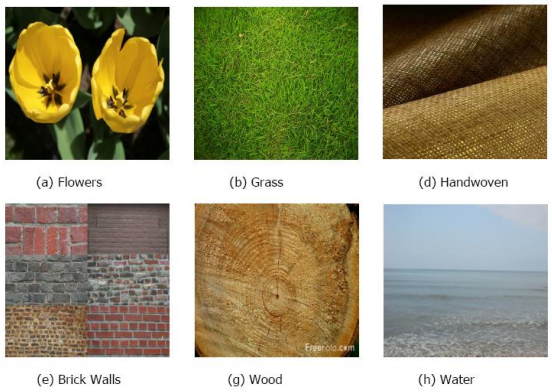

Figure 2.1: Some examples of texture features.

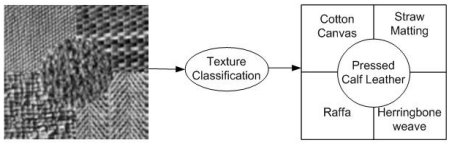

Figure 2.2: Texture classification [44].

2.3.2 Texture Analysis

Texture analysis is an essential issue in computer vision and image processing, such as in remote sensing, content based image retrieval, and so on. There are many researches about it [16, 31, 44]. Texture analysis has a four steps [31]:

- Texture extraction: In the first step of texture analysis, some basic information on the image that is useful for other steps is calculated. These information define the homogeneity or similarity between different regions of an image.

- Texture classification: An image is assigned to one of a set of predefined texture classes. Each class contains similar texture samples. For example in medical imaging, texture is used to classify Magnetic Resonance (MR) images of the brain into gray or white and regions are also classified into water, ice etc. Texture classification is an application of pattern recognition techniques, such as in medical imagery, remote sensing [17]. It relates the extraction of texture features and the design of a decision rule or classifier for classification. Textures are classified by comparing texture feature vectors extracted from the image and the reference feature vectors all of, which is basically a pattern recognition problem [35, 30, 31]. Figure 2.2 shows an example of texture classification.

- Texture segmentation: Texture segmentation is done to partition a textured image into a set of homogeneous disjointed regions so each region contains a single texture based on texture properties. Figure 2.3 shows the example of texture segmentation. The input image is divided into 3 different regions (Ground, Housing, Plants) based on texture properties [23].

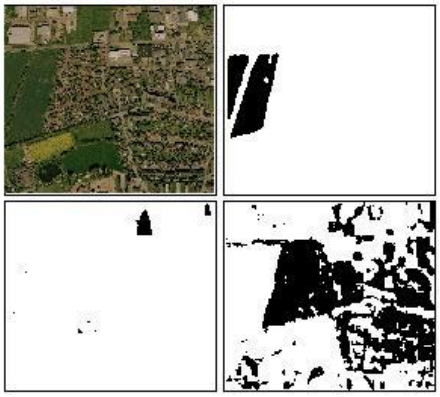

Figure 2.3: Texture segmentation [23].

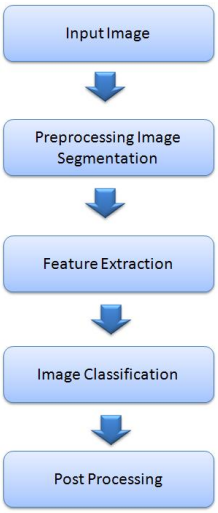

Figure 2.4: Different steps in image analysis process.

- Shape from texture: A 3-D surface shape is reconstructed from texture information on the image. Its methods provide information about shape from texture deformation features in an image. Shape from texture was produced by Gibson [35, 44].

Classification and segmentation often go together, if an image contains multiple textures, texture segmentation is required before texture classification for dividing the image into different regions, and also by classifying the individual pixels, image segmentation can be produced.

Figure 2.4 depicts the relationship between different steps in image analysis process.

2.3.3 Application of Texture

Nowadays, one of the interesting applications involves matching images is ones queried in the database; texture is useful for finding such similar pages. Texture description makes

comparisons between different textured images in the database [30]. Furthermore, the texture information is used in browsing and retrieval of large image data that pose interesting and challenging problems. An image can be viewed as a mosaic of different texture regions, and the image feature associated with these regions can be used for search and retrieval purpose. Texture analysis is used in many fields of image processing such as computer vision, pattern recognition, and content-based image retrieval.

2.3.4 Texture Feature Extraction Algorithms

Tuceryan and Jain [44] divided the different methods for feature extraction into four main categories, namely: structural, statistical, model-based, and transform domain, which are briefly explained in the following sections. Figure **??** shows different methods for texture features extraction.

2.3.4.1 Structural Methods

Structure represents a texture according to the local properties (micro-texture) and spatial organization (macro-texture) of local properties. The structural approaches provide a good symbolic description of the image, and are useful for texture generation as well as texture analysis [44]. This method is not suitable for natural textures because of the variability both of micro-texture and macro-texture and there is no clear distinction between them [35, 31].

2.3.4.2 Statistical Methods

Statistical methods represent the texture indirectly according to the non-deterministic properties that manage the distributions and relationships between the gray levels of an image. This technique is one of the first methods in machine vision [44]. By computing local features at each point in the image, and deriving a set of statistics from the distributions of the local features, statistical methods can be used to analyze the spatial distribution of gray values. Based on the number of pixels defining the local feature, statistical methods can be classified into first-order (one pixel), second-order (two pixels) and higher-order (three or more pixels) statistics. The difference between these classes is that the first-order statistics estimate properties (e.g. average and variance) of individual pixel values by waiving the spatial interaction between image pixels, but in the second-order and higher-order statistics estimate properties of two or more pixel values occurring at specific locations relative to each other. The most popular second-order statistical features for texture analysis are derived from the co-occurrence matrix [35, 31]. The approach based on multidimensional co-occurrence matrices was recently shown to outperform wavelet packets (a transform-based technique) when applied to texture classification [46]. Description of some of statistical methods are:

- First order histogram

  An image is a function $f(x, y)$ of two dimensions x and y, $x\{0, 1, ..., N_x - 1\}$ and $y = \{0, 1, ..., N_y - 1\}$. The $f(x, y)$ can take discrete values $i = \{0, 1, ..., N_g - 1\}$, N_g is the total number of intensity levels (the number of pixels in the whole image) in

the image. A histogram is a diagram which shows how many pixels of an image have a certain intensity. It has some advantages, one of them is that histograms of the image and its rotation image are the same, and another is that the size of storage place for histogram is lower than the storage size of the image [41]. The intensity level is calculated by this formula:

$$h(i) = \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} \delta(f(x,y), i), \tag{2.2}$$

where $\delta(i,j)$ is the Kronker delta function:

$$\delta(i,j) = \begin{cases} 1 & j = i \\ 0 & j \neq i. \end{cases} \tag{2.3}$$

- Second-order histogram

  In the second-order, the relationship between two pixels that usually are neighborhood is considered. In 1979, Haralick [17], defined the co-occurrence matrix as a second-order histogram statistics and it is one of the best known texture analysis methods. This method is known Gray Level Co-occurrence Matrix (GLCM). It is usefulness in applications where the space distribution of gray levels is important (e.g in radar signals), or in image analysis applications (e.g. biomedical) [28], and also it is useful for remote sensing techniques that are an important in grasping damage information caused by earthquakes [38]. There are many researches based on Haralick texture features [47, 48, 34, 42]. In the second-order, measures are used to consider the relationship between groups of two pixels (usually neighboring) in the image. It is assumed that an image is stored as a 2D array, $f(x,y)$. The spatial domains of x and y are $L_x = \{1, 2, ..., N_x\}$ as a horizontal spatial domain , $L_y = \{1, 2, \cdots, N_y\}$ as a vertical spatial domain. The $L_x \times L_y$ is the set of individual pixels and the digital image I is a function that assigns a gray level value (brightness value) of $G = \{1, 2, ..., N_g\}$ to each pixels [17]. The matrix defines the probability of joining two pixels $P_{d,\theta}(i,j)$ that have values i and j, with distance d and θ as an orientation angular. The co-occurrence matrix can be computed using two techniques. First, image pixels are separated by d and $-d$ for a given direction (θ) in four directions $(0, 45, 90, 135)$. Second, image pixels are separated by distance d in eight directions $(0, 45, 90, 135, 180, 225, 270, 315)$ [15, 31].

  This 2D histogram matrix represents the transitions between all pairs of two gray levels. The $N_{d,\theta}(i,j)$ indicates the number of transitions between two pixels whose gray levels are i, j with d pixels distant and θ as an orientation angular. A square matrix of dimension is equal to the number of intensity levels in the image, for each distance d and the orientation. The distances can be 1 and 2 pixels based on angles (θ) is chosen between 0 to 360. For classification the fine textures small values of d is required, whereas coarse textures require large values of d. A reduction in the number of intensity levels by quantizing the image to fewer levels of intensity increase the speed of computation, with losing of some textural information. Transfer pixels in the co-occurrence matrix then calculate a large number of statistical features from the matrix [31].

- Higher order: In these methods the relationships between three or more pixels are considered.

2.3.4.3 Model-Based Methods

Model based texture analysis such as Fractal model, and Markov are based on the construction of an image that can be used for describing texture and synthesizing it [44]. These methods describe an image as a probability models or as a linear combination of a set of basic functions [48].

The Fractal model is useful for modeling certain natural textures that have a statistical quality of roughness at different scales [44], and also for texture analysis and discrimination. This method has a weakness in orientation selectivity and is not useful for describing local image structures. Pixel-based models view an image as a collection of pixels, whereas region-based models view an image as a set of sub patterns. There are different types of models based on the different neighborhood systems and noise sources. These types are one-dimensional time-series models, Auto Regressive (AR), Moving Average (MA), and Auto Regressive Moving Average (ARMA). Random field models analyze spatial variations in two dimensions, global random, and local random. Global random field models treat the entire image as a realization of a random field, and local random field models assume relationships of intensities in small neighborhoods. A widely used class of local random field models are Markov models, where the conditional probability of the intensity of a given pixel depends only on the intensities of the pixels in its neighborhood (the so-called Markov neighbors) [35, 31].

2.3.4.4 Transform Domain Methods

Transform methods, such as Fourier, Gabor, and wavelet transforms represent an image in a space whose co-ordinate system has an interpretation that is closely related to the characteristics of a texture (such as frequency or size). They analyze the frequency content of the image. Methods based on Fourier transforms have a weakness in a spatial localization so they do not perform well. Gabor filters provide means for better spatial localization but their usefulness is limited in practice because there is usually no single filter resolution where one can localize a spatial structure in natural textures [35, 31]. These methods involve transforming original images by using filters and calculating the energy of the transformed images. They are based on the process of the whole image that is not good for some applications which are based on one part of the input image.

- *Multi Resolution Techniques*

  In these techniques the preservation of an image is based on a certain levels of resolution or blurring. Moreover, it zooms in and out of the underlying texture structure so the texture extraction is not affected by the size of the pixel neighborhood. The multi resolution algorithm has two steps. The first step involves the automatic extraction of the most discriminative texture features of the region of interested. In the second step, a classification that automatically identifies the various tissues is created [40].

- *Wavelet-Based transformation*
  The texture can be analyzed by methods of wavelet-based transformation. The digital information should be stored and retrieved. The wavelet compression is used for storing and retrieving. Developing an automated imaging system for the classification of tissues in medical images obtained by Computed Tomography (CT) scan. The wavelet transform are useful in image comparison, image de-noising, and image classification [40].
- *Discrete Wavelet Transform (DWT)*
  All the texture features of $n*n$ image in a region are calculated and the mean of these features is used as the region feature (Ave). The problem in DWT is that the average feature of small blocks cannot exactly describe the texture property of a region [26]. One of the way of implementing DWT is by using Discrete Wavelet Framework (DWF). Image segmentation is an important step for many image processing and computer vision algorithms. Furthermore, segmentation is useful for retrieving images from large image databases for content-based image retrieval systems. One of the variations of DWT is Discrete Wavelet Framework (DWF) which is used for textured region characterization in images. The DWF decomposition of a textured region provides a translation invariant texture description which results in a better estimation and more detailed texture characterization at region boundaries. The color and spatial feature space of the mean shift algorithm is then extended using these texture characteristics to create higher dimensional feature space for improved segmentation. A texture is characterized by a set of median values of energy estimated in a local window at the output of the corresponding filter bank. The energy in a local window can be calculated using coefficients of DWF decompositions (LL, LH, HL, and HH).where the energy is defined as the square of the coefficients. The advantage of a using median filter is that it preserves the energy associated with texture between regions. The sub-bands at the output of filter bank are approximate, horizontal, vertical and diagonal components of the input image signal. Most of the texture information is contained in LH and HL sub-bands, only these decomposition coefficients are used to obtain texture features. A pixel in a textured region can be classified into one of four texture categories based on texture orientation. The texture categories are vertical, horizontal, smooth (not enough energy in any orientation), and complex (no dominant orientation). Texture feature extraction consists of two steps. First, the energy of LH and HL sub-bands are classified into two categories (0 and 1) using K-means clustering algorithm. Second, a further classification is made using a combination of two categories in each sub-band LH and HL. A pixel is classified as smooth if its category is 0 in both LH and HL sub-bands. A pixel is classified as vertical if its category is 0 in LH, and 1 in HL subbands. Similarly, a pixel is classified horizontal if its category is 1 in LH, and 0 in HL sub-bands. Finally, a pixel is classified as complex if its category is 1 in both the LH and HL sub-bands. The goal is to characterize image pixels by using these texture features and to use them to extend the mean shift feature space to obtain better segmentation [36].

  - *Ridgelet-Based transformation*
    The Ridgelet transform derives information of an image that is based on multiple radial directions not just in vertical and horizontal frequency domains. First, order statistics can be calculated on the directional then texture descriptors that can be used in the classification of texture are provided by applying a 1D wavelet transform. In the Computed Tomography (CT) medical image the ridgelet-based algorithm is more useful than others [40].

- Fourier Transform

  The Fourier Transform (FT) is in the signal processing methods and maps a signal into its component frequencies or decomposes an image into its sine and cosine components. It is useful in image compression, image filtering, etc. It is defined by either

$$f(k,l) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) e^{-i2\pi(\frac{ki}{N} + \frac{lj}{N})}, \tag{2.4}$$

  where $f(i,j)$ is the image in the spatial domain and $K = 0, 1, ..., N-1$,

  or

$$F_p(w) = \int p(t) e^{-jwt} dt, \tag{2.5}$$

  where w is the angular frequency and $w = 2\pi f$ in radians/s and f is the frequency that $f = (1/t)$, t is the time, j is the complex variable and the $p(t)$ is a continuous signal in time, and $e^{-jwt} = \cos(wt) j \sin(wt)$ is the frequency components in $x(t)$ [33].

  - Discrete Fourier Transform
    The DFT decomposes image into components of different frequencies and its complexity is $O(N^2)$. The weakness of DFT is that it is slow.
  - Fast Fourier Transform
    The Fast Fourier Transform has the same result with DFT but is much faster. It calculates one dimension of DFT. The complexity of FFT is $O(N \log N)$.

- Gabor

  The Gabor Filters (GF) by Dennis Gabor [13] are linear filters that are used in image analysis applications, such as texture classification, texture segmentation, image recognition, edge detection, image representation, etc. The GF extract information is based on time and frequency. These are useful for feature extraction from 2D images in texture analysis [35, 20]. The GF is defined according to this formula:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp(-\frac{\acute{x}^2 + \gamma^2 \acute{y}^2}{2\sigma^2}) \cos(2\pi \frac{\acute{x}}{\lambda} + \psi) \tag{2.6}$$

  where

$$\acute{x} = x \cos\theta + y \sin\theta, \tag{2.7}$$

$$\acute{y} = -x \sin\theta + y \cos\theta. \tag{2.8}$$

The λ represents the wavelength of the cosine, θ is the orientation, ψ is the phase offset, σ is the sigma of the Gaussian envelope, and γ is the spatial aspect ratio.

The Gabor features are calculated by the below formula and (x, y) is the spatial coordinate, f as a frequency, and θ for orientation:

$$\begin{aligned} r_\xi(x, y; f, \theta) &= \Psi(x, y; f, \theta) * \xi(x, y) \\ &= \int\int \Psi(x - x_\tau, y - y_\tau; f, \Theta)\xi(x_\tau, y_\tau)\, dx_\tau\, dy_\tau. \end{aligned} \tag{2.9}$$

Calculation texture features by Gabor has some difficulties, such as for determining the size of the Gabor filter window and the number of Gabor channels at the same radial frequency [48].

Compared with the Gabor transform, the Wavelet transform feature has several advantages and are suitable for texture analysis [31]:

- varying the spatial resolution allows it to represent textures on the most suitable scale,
- There is a wide range of choices for the Wavelet function, so one is able to choose the wavelet best suited to texture analysis in a specific application.

Furthermore, DWT is localized in both time and frequency and Fourier transform is localized in frequency.

A texture description with Wavelet methods is done by filtering the image with a bank of filters, each filter having a specific frequency (and orientation), then texture features are extracted from the filtered images. For image with large dimension, often many scales and orientations are needed [35]. These Wavelet advantages make it attractive for texture segmentation.

2.3.5 Rules for Choosing Texture Extraction Algorithms

Each of these categories has some algorithms for texture feature extraction. To select these algorithms some characteristics should first be considered, these characteristics are as follow [35]:

- Illumination (gray-scale) invariance: The sensitivity of algorithm to change in gray scale. This aspect is an important when the lighting condition in industrial machine vision may be unstable.
- Rotation invariance: Does the texture of images change if we change the rotation of the images.
- Robustness in front of noise: The robustness ability of the algorithm is in the noisy environment which has an effect on input image.
- Computational Complexity: Many algorithms are computationally intensive, for example in retrieval applications for large databases

- Generatively: Can the algorithm facilitate texture synthesis, i.e. by regenerating the texture that was captured using the algorithm.
- Popularity: Which of them are more popular and more practical.
- Easy to implement: The algorithm should be simple to implement.

Complexities of DWT, and GLCM are $O(n)$, and fast Fourier transform and gabor are $O(nlogn)$. I have chosen GLCM, based on its some characteristics as follows. Nowadays, the most common way to extract texture features is GLCM, it has been used in many applications, such as in content based image retrieval, biomedical, etc. Furthermore, GLCMs of an original image is the same with GLCMs of its rotation, hence it has a rotation invariance character. In the section 3.8, we discussed about the comparison between GLCM and other algorithms.

2.3.6 Statistical Algorithms for Texture Extraction

- *First-order histogram based features*: This method provides the 1D histogram of an image based on its gray level. The histogram is simply a summery of the statistical information about the image. The probable density ($p(i)$) of occurrence of the intensity levels is calculated by dividing the values $h(i)$ in the total number of pixels in the $N_x \times N_y$ image.

$$p(i) = h(i)/N_x N_y, i = \{0, 1, ..., N_g - 1\}. \tag{2.10}$$

  The histogram defines the characteristics of the image, for example, a narrowly distributed histogram indicated the low-contrast image. A bimodal histogram often suggests that the image contained an object with a narrow intensity range against a background of differing intensity [31].

  The features that can be extracted are:

  - Mean: The mean defines the average level of intensity of the image or texture

$$\mu = \sum_{i=0}^{N_g-1} ip(i). \tag{2.11}$$

  - Variance: Which defines the variation of intensity around the mean

$$\sigma^2 = \sum_{i=0}^{N_g-1} (i-\mu)^2 p(i). \tag{2.12}$$

  - Skewness: It defines the symmetry.

$$\mu^3 = \sigma^{-3} \sum_{i=0}^{N_g-1} (i-\mu)^3 p(i). \tag{2.13}$$

$$\text{Skewness} = \begin{cases} \mu^3 < 0 & \rightarrow \quad \text{Histogram is below the mean} \\ \mu^3 = 0 & \rightarrow \quad \text{Histogram is symmetrical to the mean} \\ \mu^3 > 0 & \rightarrow \quad \text{Histogram is above the mean} \end{cases} \tag{2.14}$$

– Kurtosis: This is a measure of the flatness of the histogram

$$\mu^4 = \sigma^{-4} \sum_{i=0}^{N_g-1} ((i-\mu)^4 p(i)) - 3. \tag{2.15}$$

– Energy: That returns the sum of squared elements

$$E = \sum_{i=0}^{N_g-1} [p(i)]^2. \tag{2.16}$$

– Entropy:

$$H = -\sum_{i=0}^{N_g-1} p(i) \log_2[p(i)]. \tag{2.17}$$

The other features which can be archived of the histogram are the maximum, minimum, median, and the range. The information of this histogram is used as features for texture segmentation. The module that a measurement of histogram's information is calculated by:

$$I_{N_y H}(x,y) = \sum_{i=0}^{N_g-1} \frac{h(i) - N_x/G}{\sqrt{h(i)[1-p(i)] + N_x/N_g(1-1/N_g)}}. \tag{2.18}$$

The result of this technique is simple but texture cannot be completely characterized. This method is not useful for a large class of texture [31], and the other weakness of this method is that the histogram of two different images that have the same gray value for different pixels are equals. The complexity of this method for the $N_x \times N_y$ image is $O(N_x * N_y)$.

Example 3.1: Figure 2.5 shows two gray level matrix of size 5×5 and their histograms. Although, images are not same but their histograms is the same. That is a limitation of technique.

- *Gray-Level Co-occurrence matrix*

  The Gray-Level Co-occurrence Matrix (GLCM) is based on the extraction of a gray-scale image. It considers the relationship between two neighboring pixels, the first pixel is known as a reference and the second is known as a neighbor pixel [19]. The GLCM is a square matrix with N_g dimension, where N_g equals the number of gray levels in the image. Each element of the matrix is the number of occurrence of the pair of pixel with value i and a pixels with value j which are at distance d [6, 28, 12].

  The measuring of texture involves the following steps:

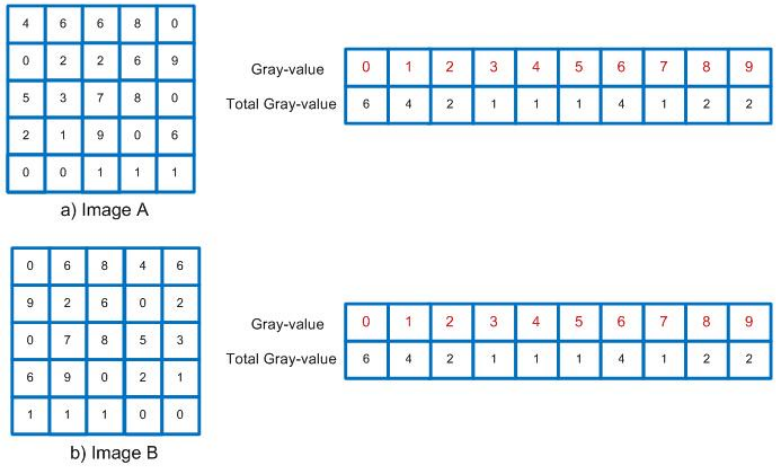

Figure 2.5: Example of first-order methods.

– Make the GLCM symmetrical

– Calculate the probability of each combination, the probability is calculated:

$$P_{(i,j,d,\theta^0)} = \#\{((k,l),(m,n)) \quad \in (L_r \times L_c) \times (L_r \times L_c)| \\ (k-m),(l-n) \in \{-d,0,d\}|I(k,l)=i, I(m,n)=j, |\angle((k,l),(m,n)) = \theta\} \tag{2.19}$$

And $p(i,j)$ is the element (i,j)th of the normalized co-occurrence matrix

$$p_{d,\theta}(i,j) = \frac{P_{(i,j,d,\theta^0)}}{\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} P_{(i,j,d,\theta^0)}} \tag{2.20}$$

If the co-occurrence matrix is symmetric then $p(i,j) = (p(i,j) + p(i,j)^T)/2$ that T indicates the transpose matrix and θ will be 0, 45, 90 and 135 [4].

– Calculated the texture features.

Haralick et al [17] defined 14 texture features, these features contain the information about the image such as homogeneity, contrast, the complexity of the image, and etc. They are used in many applications such as biological applications and image retrieval.

This adjacency can occur in four directions based on the angle, horizontal, vertical, right diagonal, and left diagonal. Figure 2.6 shows these directions.

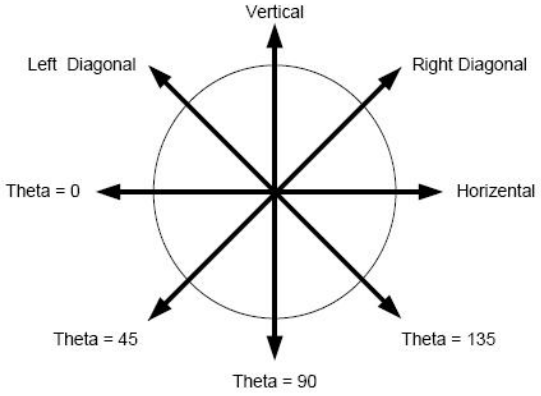

Figure 2.6: Diagram of angles, the Haralick texture features are calculated in each of these directions.

The following equations are needed for calculating Haralick texture feature.

$$p_x(i) = \sum_{j=1}^{N_g} p_{d,\theta}(i,j), \tag{2.21a}$$

$$p_y(j) = \sum_{i=1}^{N_g} p_{d,\theta}(i,j), \tag{2.21b}$$

$$p_{x+y}(k) = \sum_{i=1}^{N_g}\sum_{j=0}^{N_g} p_{d,\theta}(i,j), \quad k=\{2,3,...,2N_g\}, \quad k=i+j, \tag{2.21c}$$

$$p_{x-y}(k) = \sum_{i=1}^{N_g}\sum_{j=0}^{N_g} p_{d,\theta}(i,j), \quad k=\{0,1,...,N_g\}, \quad k=|i-j|. \tag{2.21d}$$

Haralick Texture Features:

With this method, 14 texture features are taken for each image. The features are as follows:

1. Angular Second Moment (ASM)
   ASM also known as uniformity or energy, measures the image homogeneity. ASM is high when pixels are very similar.

$$f_1 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_{d,\theta}(i,j)^2. \tag{2.22}$$

2. Contrast (CON)
   Contrast is a measure of intensity or gray-level variations between the reference pixel and its neighbor. The visual perception is the difference in appearance of two or more parts of a field seen simultaneously or successively.

$$f_2 = \sum_{n=0}^{N_g-1} n^2 \{ \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_{d,\theta}(i,j) \} \quad |i-j| = n. \tag{2.23}$$

3. Correlation (COR)
   Correlation calculates the linear dependency of the gray level values in the co-occurrence matrix [29]. It shows how the reference pixel is related to its neighbor.

$$f_3 = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (ij) p_{d,\theta}(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{2.24}$$

   Where:
   μ_x, μ_y, σ_x, and σ_y are the means and standard deviations of p_x and p_y.

4. Sum of Squares: Variance
   This is a measure of gray tone variance.

$$f_4 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i-\mu)^2 p_{d,\theta}(i,j). \tag{2.25}$$

5. Inverse Difference Moment (IDM)
   IDM also sometimes called homogeneity, measures the local homogeneity of a digital image. IDM returns the measures of the closeness of the distribution of the GLCM elements to the GLCM diagonal.

$$f_5 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \frac{1}{1+(i-j)^2} p_{d,\theta}(i,j) \tag{2.26}$$

6. Sum Average (mean)

$$f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i) \tag{2.27}$$

7. Sum Variance

$$f_7 = \sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i) \tag{2.28}$$

8. Sum Entropy

$$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log p_{x+y}(i) \tag{2.29}$$

If the probability equals zero then the $\log(0)$ is not defined. To prevent this problem, it is recommended to use $\log(p + \varepsilon)$ that ε is an arbitrarily small positive constant, instead of $\log(p)$.

9. Entropy (ENT)
Entropy shows the amount of information of the image that are needed for image compression.

$$f_9 = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_{d,\theta}(i,j) \log(p_{d,\theta}(i,j)) \tag{2.30}$$

The high entropy image has a great contrast from one pixel to the its neighbor and cannot be compressed as a low entropy image which has a low contrast (a lot of amount of pixels have the same or similar value) [6].

10. Difference Variance

$$f_{10} = \text{variance of } p_{x-y} \tag{2.31}$$

11. Difference Entropy

$$f_{11} = -\sum_{i=0}^{N_{g-1}} p_{x-y}(i) \log p_{x-y}(i) \tag{2.32}$$

12. Information Measures of Correlation 1

$$f_{12} = \frac{HXY - HXY1}{\max(HX, HY)} \tag{2.33}$$

13. Information Measures of Correlation 2

$$f_{13} = (1 - \exp[-2.0(HXY@ - HXY)])^{1/2} \tag{2.34}$$

where:

$$HXY = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_{d,\theta}(i,j)\log(p_{d,\theta}(i,j)) \tag{2.35}$$

HX and HY are entropies of p_x and p_y

$$HXY1 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_{d,\theta}(i,j)\log\left(p_x(i)p_y(j)\right) \tag{2.36}$$

$$HXY2 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_x(i)p_y(j)\log p_x(i)p_y(j) \tag{2.37}$$

14. Maximal Correlation Coefficient

$$f_{14} = (\text{Second largest eigenvalue of } Q)^{1/2} \tag{2.38}$$

where

$$Q(i,j) = \sum_k \frac{p_{d,\theta}(i,k)p_{d,\theta}(j,k)}{p_{x(i)}p_{y(k)}} \tag{2.39}$$

The variance is a measure of the dispersion of the values around the mean, it is similar to the entropy. It is calculated by these formulas [15]: The complexity of Haralick for an $N \times N$ image is $O(N^2)$.

Example 3.2: A 4×4 image with four gray level values $0 - 3$ is assumed.

The image is normalized as follows.

$$PH = \left\{ \begin{array}{cccc} 0.125 & 0.125 & 0.042 & 0.042 \\ 0.042 & 0.042 & 0.083 & 0.083 \\ 0.083 & 0.083 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.125 & 0.125 \end{array} \right\}.$$

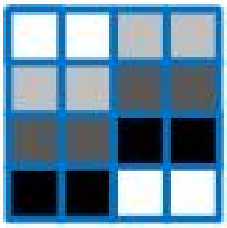

Figure 2.7: An image of size 4×4.

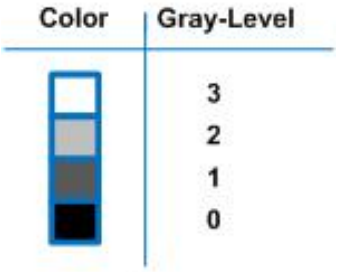

Figure 2.8: Gray tone color

The mean is calculated as follows.

$$\begin{array}{rl} & 1 * (0.125 + 0.125) \\ + & 2 * (0.042 + 0.042 + 0.083 + 0.083) \\ + & 3 * (0.083 + 0.083) \\ + & 4 * (0.125 + 0.125) \\ = & 1.332 \end{array} \tag{2.40}$$

For extracting texture features first the GLCM is calculated. Table 2.1 depicts the construction of the GLCM for this example. Each element (i, j) of the matrix shows the total number of times that two gray tones of element i and j is occurred based on a function of angle adjacent to each other.

The boundary of distance is calculated:

$$d((k, l), (m, n)) = \max\{|\text{k - m}|, |\text{l - n}|\}.$$

The GLCM and Haralick features can be calculated using two techniques. In the first technique, the GLCM and Haralick features are calculated for each direction individually, and then textures features of the input image are calculated based on the features of each direction. In the second technique, the GLCM and features are calculated for all of directions at the same time.

In the first approach, the GLCM and texture features are calculated for each direction by assuming that distance (d) is equal to 1.

– **Horizontal GLCM ($\theta = 0°$)**

i, j	1	2	3	4
1	$\#(0,0)$	$\#(0,1)$	$\#(0,2)$	$\#(0,3)$
2	$\#(1,0)$	$\#(1,1)$	$\#(1,2)$	$\#(1,3)$
3	$\#(2,0)$	$\#(2,1)$	$\#(2,2)$	$\#(2,3)$
4	$\#(3,0)$	$\#(3,1)$	$\#(3,2)$	$\#(3,3)$

Table 2.1: Construction of co-occurrence matrix.

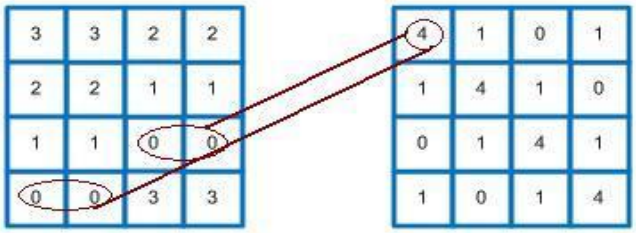

Figure 2.9: The symmetrical horizontal GLCM.

$$\begin{cases} \text{k - m} = 0, \\ |\text{l - n}| = \text{d}. \end{cases}$$

* **Symmetrical Horizontal GLCM:**
Each element of the GLCM, $p(i, j)$, is a number of times that two pixels with gray-tone i, and j are neighborhood in distance d, and directions θ. Figure 2.9 shows how the symmetrical horizontal GLCM is calculated. The value in $(0, 0)$ is the number of times that two pixels with gray-tone 0 are neighborhood.

$$PH = \begin{Bmatrix} 12 & 1 & 0 & 2 \\ 1 & 8 & 1 & 0 \\ 0 & 1 & 12 & 2 \\ 2 & 0 & 2 & 16 \end{Bmatrix}.$$

* **Normalized Symmetrical Horizontal GLCM:**
The Normalization of GLCM: Each element of GLCM contains a probability that is the value of each element divided by the total value of all of them. The total gray-value is 24.

$$PH = \begin{Bmatrix} 0.2 & 0.017 & 0.000 & 0.033 \\ 0.017 & 0.133 & 0.017 & 0.000 \\ 0.000 & 0.017 & 0.2 & 0.033 \\ 0.033 & 0.000 & 0.033 & 0.267 \end{Bmatrix}.$$

The ASM is: The ASM is 0.175.
The mean is:

$$\begin{aligned} & 1 * (0.2 + 0.017 + 0.033) \\ + \; & 2 * (0.017 + 0.133 + 0.017) \\ + \; & 3 * (0.017 + 0.2 + 0.033) \\ + \; & 4 * (0.033 + 0.033 + 0.267) \\ = \; & 3.338 \end{aligned} \tag{2.41}$$

– **Right Diagonal GLCM ($\theta = 45°$)**

$$\begin{cases} \text{k - m = d, -d,} \\ \text{|l - n |} = \text{-d, d.} \end{cases}$$

∗ **Symmetrical GLCM:**

$$PH = \begin{Bmatrix} 6 & 4 & 0 & 3 \\ 4 & 2 & 4 & 0 \\ 0 & 4 & 6 & 3 \\ 3 & 0 & 3 & 8 \end{Bmatrix}.$$

∗ **Normalized Symmetrical Left Diagonal GLCM:**
The total gray-value is 18.

$$NPLD = \begin{Bmatrix} 0.12 & 0.08 & 0.000 & 0.06 \\ 0.08 & 0.04 & 0.08 & 0.000 \\ 0.000 & 0.08 & 0.12 & 0.06 \\ 0.06 & 0.000 & 0.06 & 0.16 \end{Bmatrix}.$$

The ASM is:

$$\begin{aligned} & 0.12^2 + 0.08^2 + 0.000^2 + 0.06^2 \\ + \; & 0.08^2 + 0.04^2 + 0.08^2 + 0.000^2 \\ + \; & 0.000^2 + 0.08^2 + 0.12^2 + 0.06^2 \\ + \; & 0.06^2 + 0.000^2 + 0.06^2 + 0.16^2 \\ = \; & 0.096 \end{aligned} \tag{2.42}$$

The mean is:

$$\begin{aligned} & 1 * (0.12 + 0.08 + 0.00 + 0.06) \\ + \; & 2 * (0.08 + 0.04 + 0.08 + 0.00) \\ + \; & 3 * (0.00 + 0.08 + 0.12 + 0.06) \\ + \; & 4 * (0.06 + 0.00 + 0.06 + 0.16) \\ = \; & 3.12 \end{aligned} \tag{2.43}$$

– **Vertical GLCM ($\theta = 90°$)**

$$\begin{cases} \text{k - m = d, -d,} \\ \text{|l - n |} = \text{-d, d.} \end{cases}$$

∗ **Symmetrical Vertical GLCM:**

$$NPLD = \left\{ \begin{array}{cccc} 0.12 & 0.08 & 0.000 & 0.06 \\ 0.08 & 0.04 & 0.08 & 0.000 \\ 0.000 & 0.08 & 0.12 & 0.06 \\ 0.06 & 0.000 & 0.06 & 0.16 \end{array} \right\}.$$

$$PV = \left\{ \begin{array}{cccc} 6 & 6 & 0 & 3 \\ 6 & 0 & 6 & 0 \\ 0 & 6 & 6 & 3 \\ 3 & 0 & 3 & 12 \end{array} \right\}.$$

∗ **Normalized Symmetrical Vertical GLCM:**
The total gray-value is 24.

$$NPV = \left\{ \begin{array}{cccc} 0.1 & 0.1 & 0.00 & 0.05 \\ 0.1 & 0.00 & 0.1 & 0.00 \\ 0.00 & 0.1 & 0.1 & 0.05 \\ 0.05 & 0.00 & 0.05 & 0.2 \end{array} \right\}.$$

The ASM is:

$$\begin{array}{rl} & 0.1^2 + 0.1^2 + 0.00^2 + 0.05^2 \\ + & 0.1^2 + 0.00^2 + 0.1^2 + 0.00^2 \\ + & 0.00^2 + 0.1^2 + 0.1^2 + 0.05^2 \\ + & 0.05^2 + 0.00^2 + 0.05^2 + 0.2^2 \\ = & 0.11 \end{array} \tag{2.44}$$

The mean is:

$$\begin{array}{rl} & 1 * (0.1 + 0.1 + 0.00 + 0.05) \\ + & 2 * (0.1 + 0.00 + 0.1 + 0.00) \\ + & 3 * (0.00 + 0.1 + 0.1 + 0.05) \\ + & 4 * (0.05 + 0.00 + 0.05 + 0.2) \\ = & 3.2 \end{array} \tag{2.45}$$

– **Left Diagonal GLCM** ($\theta = 135°$)

$$\left\{ \begin{array}{l} \text{k - m = d, -d,} \\ \text{|l - n |} = \text{-d, d.} \end{array} \right.$$

∗ **Symmetrical GLCM:**

$$PLD = \left\{ \begin{array}{cccc} 4 & 4 & 1 & 3 \\ 4 & 0 & 4 & 2 \\ 1 & 4 & 4 & 3 \\ 3 & 2 & 3 & 8 \end{array} \right\}.$$

* **Normalized Symmetrical Left Diagonal GLCM:**
The total gray-value is 18.

$$PLD = \left\{ \begin{array}{cccc} 0.08 & 0.08 & 0.02 & 0.06 \\ 0.08 & 0.00 & 0.08 & 0.04 \\ 0.02 & 0.08 & 0.08 & 0.06 \\ 0.06 & 0.04 & 0.06 & 0.16 \end{array} \right\}.$$

The ASM is:

$$\begin{array}{rl} & 0.08^2 + 0.08^2 + 0.02^2 + 0.06^2 \\ + & 0.08^2 + 0.00^2 + 0.08^2 + 0.04^2 \\ + & 0.02^2 + 0.08^2 + 0.08^2 + 0.06^2 \\ + & 0.06^2 + 0.04^2 + 0.06^2 + 0.16^2 \\ = & 0.082 \end{array} \quad (2.46)$$

The mean is:

$$\begin{array}{rl} & 1 * (0.08 + 0.08 + 0.02 + 0.06) \\ + & 2 * (0.08 + 0.00 + 0.08 + 0.04) \\ + & 3 * (0.02 + 0.08 + 0.08 + 0.06) \\ + & 4 * (0.06 + 0.04 + 0.06 + 0.16) \\ = & 3.28 \end{array} \quad (2.47)$$

In the second technique the GLCM is calculated as follows by assuming d equals 1 is that:

- **Symmetrical GLCM:** The GLCM is calculated for four directions

$$PT = \left\{ \begin{array}{cccc} 28 & 9 & 1 & 11 \\ 15 & 10 & 15 & 2 \\ 1 & 15 & 28 & 11 \\ 11 & 2 & 11 & 44 \end{array} \right\}.$$

- **Normalized Symmetrical Left Diagonal GLCM:** The total gray-value is 76.

$$PT = \left\{ \begin{array}{cccc} 0.127 & 0.068 & 0.005 & 0.05 \\ 0.068 & 0.045 & 0.068 & 0.009 \\ 0.005 & 0.068 & 0.127 & 0.05 \\ 0.05 & 0.009 & 0.05 & 0.2 \end{array} \right\}.$$

The ASM is: The ASM is 0.103 that near sum of all ASMs values of each directions.

The mean is: The mean is 3.234, with comparing this mean and the average of means for all angle, we can conclude that the mean of the input image is the average of means each directions.

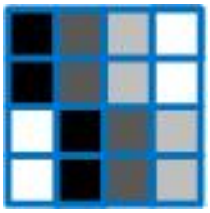

Figure 2.10: 4×4 rotated image.

Example 3.3 : One of the characters of GLCM is that with rotation the input image, its GLCM does not change. If the image is rotated, the gray-scale view of image is, Figure 2.10 shows the gray level of this image.

The GLCM matrix is:

– **GLCM:**

$$PH = \begin{Bmatrix} 28 & 15 & 1 & 11 \\ 15 & 10 & 15 & 2 \\ 1 & 15 & 28 & 11 \\ 11 & 2 & 11 & 44 \end{Bmatrix}.$$

– **Normalized GLCM:**

$$PH = \begin{Bmatrix} 0.127 & 0.068 & 0.005 & 0.05 \\ 0.068 & 0.045 & 0.068 & 0.009 \\ 0.005 & 0.068 & 0.127 & 0.05 \\ 0.05 & 0.009 & 0.05 & 0.2 \end{Bmatrix}.$$

The ASM is: The ASM is 0.103.

The mean is: The mean is 3.234

We can conclude the GLCM and texture features of the input image and its rotated image are the same.

2.3.7 Comparison between Co-occurrence and others Algorithms

A vast body of work on comparing texture features algorithms exists. Weszka et al [47] applied texture features of Haralick and Fourier on photographs of nine Terrain types (Lake, Marsh, Orchard, Railroad, Scrub, Suburb, Swamp, Urban, and Woods) for texture classification, their results shows haralick features have better performance than Fourier. Ohaniand et al [34] compared performance of features of Markov, Gabor, Fractal, and GLCM in recognizing classes of visual textures. They concluded GLCM has a better performance than others. du Buf et al [48] compared features of different algorithms for image segmentation and their results show GLCM has a better performance. On

the other hand, GLCM has some weakness. In feature based segmentation application, the classification is performed in the feature space constructed by entropy, correlation, energy, contrast and homogeneity feature [18]. Furthermore, for small image size, and gray-tone values, GLCM is computed as a spareness. In total, GLCM is suitable for gray-tone values more than 32.

2.3.8 Computational Overhead of Co-occurrence Processing

The overall computational complexity time is computation time of calculation the GLCM, normalization of the GLCM, and calculation of texture features. Most of the time is spent for calculation of GLCM. There are different methods for decreasing the GLCM time consumption. In one method the image is represented by four or five bits instead of eight bits that makes to reduce the size of GLCM but it makes to remove some information about the image. Another method is that to reduce the size of GLCM by storing just non-zero values. Clausi and Jernigant [9] describe the gray Level Co-occurrence Linked List (GLCLL). In GLCLL just non-zero values of GLCM are stored in a linked list, and each linked list node contains the two co-occuring gray-values, the co-occurrence probability of these two pair gray-values, and a link to the next node. when a new pair (i, j) comes, first there is a search for finding i, if it is found then there is a search for j. If there is a (i, j) in the list, their probability is increased, else the new node is added to the list. The GLCLL increases the calculation time. In 2001, Clausi and ZhaoThe [10] Gray Level Co-occurrence Hybrid Structure (GLCHS) that is based on an integrated hash table and linked list. Each node of the linked list includes two integer elements to store the gray-value pairs and two pointers to the previous and the next node. In the hash table, one element stores the probability of the GLCM and another stores the linked list pointer. Access to the hash table is provided by using (i, j). Each entry in the hash table has a pointer. A null pointer indicates that a particular co-occurring pair (i, j) does not have a representative node. Each new node inserted at the end of the linked list and its gray level values would be set. If the pointer is not null, then the probability of the existing node on the linked list is increased. The hash table allows rapid access to an (i, j). GLCHS is faster than GLCLL, and is useful for large image but it results increased a complexity of implementation due to a two dimensional hash table with longer linked list [10, 9]. In 2002, Clausi and Zhao [11] present new matrix of GLCM, the gray Integrated Algorithm (GLCIA) based on the combination between the gray Level Co-occurrence Hybrid Structure (GLCHS) and the gray Level Co-occurrence Hybrid Histogram (GLCHH).

3 Related Work

The co-occurrence matrix is a statistical model that is useful in a variety of image analysis applications, such as in biomedical [42], remote sensing [2], industrial defect detection systems [27], etc. FPGAs are reconfigurable hardware devices and have ability to execute many complex computations in parallel, these abilities enable a hardware system dedicated to performing fast co-occurrence matrix computations [4]. The Very Large Scale Integration (VLSI) architectures could be considered as competitive options [1] but they are not reconfigurable, and also have a high development cost and time consuming development process. As the calculation of GLCM is important, many researchers have focused to accelerate this important kernel such as [24, 5, 3, 4, 29, 42, 43].

An FPGA-based system for the computation of two GLCM features, namely mean and contrast, has been proposed by Heikkinen and Vuorimaa without actually computing the GLCMs [24]. Tahir et al. [29, 42, 43] have presented architectures that calculates the GLCM of multi-spectral images. In their first design GLCMs are calculated in software and features in hardware, and in their later works, the GLCM is calculated by one FPGA and the GLCM features is calculated by a second core after that is after programmed onto the FPGA. Uses the second core makes a time overhead for reprogramming the FPGA, that have affected on the overall feature extraction performance. Iakovidis et al. [5, 3, 4] presented some same and different designs. Their first design [5] is the same by Tahir et al. [29], their system calculates the GLCMs and features in hardware but some part of features computation are done in software (divion part). Their second design [5, 3] provides more efficient calculation of GLCMs, but it cannot calculate any GLCM features in hardware. In addition, the transfer of GLCMs over the PCI bus incurs a significant performance overhead that can be elimination for real-time video texture analysis. Their third design [4],was able of GLCM features calculation in hardware, but employed data redundancy in order to achieve high processing throughput.

Tahir et al. [43] presented an FPGA based co-processor for GLCM texture features measurement. They used Handel-C, which is developed C programming language for hardware design, for implementing of GLCM texture features on FPGA. They calculated seven of the texture features, mean, contrast, dissimilarity, angular second moment, variance, correlations and entropy. The reconfigurable computing platform that they used is the Celoxica RC1000-PP PCI based FPGA development board that equipped by a Xilinx XCV2000E Virtex FPGA. XCV2000E Virtex has a $19,200$ slices and $655,360$ bits of block RAM, and four 2 MB SRAM banks.Figure 3.1 indicates the system architecture for calculation GLCM texture features as presented in [43]. As mentioned before, texture has an important part in the classification of medical images, different regions in tissue section images can be classified as cancer and normal using texture, for this reason, they divided each input image to four sub-regions of size $N \times N$, and the best value for the N is 128 to have good localization and accurate measurements of texture

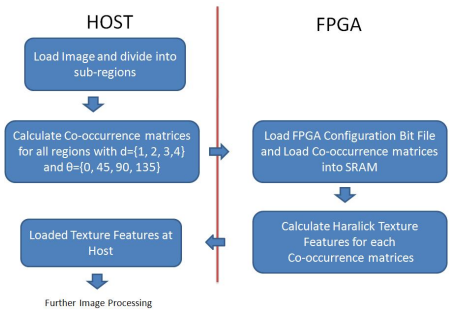

Figure 3.1: System architecture for calculation GLCM texture features [43].

features [37]. The GLCM for each sub-region is calculated for 4 distances $d = \{1, 2, 3, 4\}$ and 4 directions $\theta = 0, 45, 90, 135$ at host, then all results are loaded into 4 different SRAM banks in Celoxica RC1000-PP PCI. FPGA reads the GLCM results from SRAM banks and calculates features. Features are stored into bank0 of SRAM for image processing (segmentation, classification, and etc). THe input image is stored into memory bank 0 for acceding to the reference pixels and the same sub-region are stored into the other 3 banks of SRAM to access 4 neighbors pixels of reference pixel then 16 GLCM are calculated in parallel for reference pixel. Each element of GLCM is updates the number of occurrence of pixel. Then all GLCM are normalized in parallel and the result of calculation features are stored into bank 0 of SRAM for further image processing such as image classification, image segmentation, and etc. These calculations are done for each pixel of the input image. After the calculation for all pixels in one sub-region is done, the next sub-region is loaded and these processes are repeated. Then seven of Haralick texture features are calculated. Calculation of features has two steps, in the first step, mean, contrast, dissimilarity, and entropy are calculated, and in the second step angular second moment, variance, and correlation depend upon the value of mean are calculated. Figure 3.2 shows the block diagram of extraction Haralick features. There are five Processing Units (PUs), the first four PU include adders and multipliers, and calculate seven texture features at distance d for different directions θ, PEs are executed in parallel. The final PU includes adders and shift registers and calculates the average of each feature that is calculated at distance d for different angles θ, and results are stored in SRAM bank 0. These processes will repeat for another distance. The feature calculation operation has two steps, in the first step, mean, contrast, entropy, and dissimilarity are calculated into four different Processing Elements (PEs). PEs contain multipliers and adders that execute in parallel. Furthermore, for increasing the computation speed, $(i - j)$ and $(i - j)^2$ are pre-computed and stored in ROM, and also log tables in block RAM are used for the calculation of the log function in the entropy. In the second step, angular second moment, variance, and correlation are calculated. In the computation, the real number arithmetic and fixed point number are used.

The Handel-C is a high level language for implementing algorithms in hardware, it has a parallel composition keyword (par) to allow statement in a block to be executed in parallel [29], the PEs are executed in parallel by using this keyword. The output from

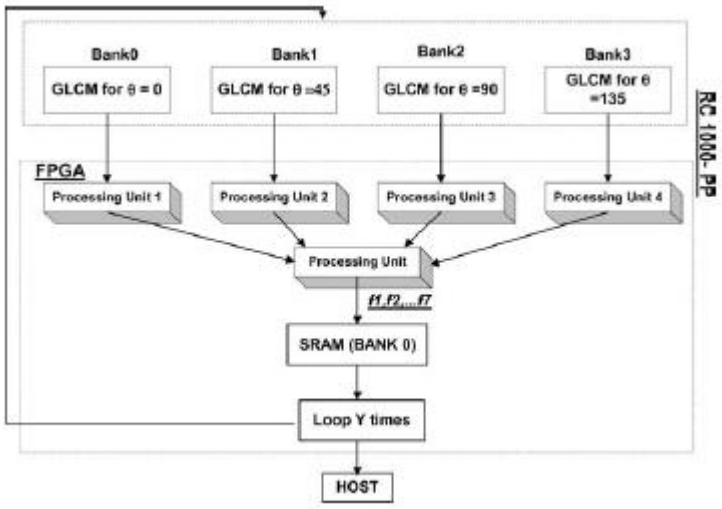

Figure 3.2: Block diagram of extraction Haralick features [43].

Handel-C is used to create the configuration data for the FPGA Celoxica DK1 is used to compiles the C program into synchronous hardware [22]. Based on their experiment, the performance of implementation on FPGA is 7 times faster than implementation on Pentium 4 with 2400 MHz clock, even the PC has a clock speed more 50 times faster than clock speed of FPGA [29]. In A 16 bit integer for the GLCM, a 32 bit floating point for the normalization of the GLCM are used, and in FPGA a 14 bit fixed point number for the GLCM and 24, 20, 16 bit fixed point numbers for the normalization of GLCM. Their results show that the performance of FPGA is approximately 9 times faster than Pentium 4, and also the speed of FPGA are independent of the image size. FPGA on their design executes one sub-region at a time and the rest of sub-regions are looped. They used pipelining and parallelism for implementation. A later work by Tahir et al. [42] presents an FPGA based coprocessor for GLCM and Haralick texture features and their applications in prostate cancer classification. Figure 3.3 indicates their algorithms. Their target device is the Celoxica RC1000-PP PCI based FPGA development board that equipped by a Xilinx XCV2000E Virtex FPGA. XCV2000E Virtex has a $19,200$ slices and $655,360$ bits of block RAM, and four banks of SRAM with 2MB for each of them. The system model of their design is shown in Figure 3.4. In contrast with there's another research [29], they calculate GLCM and texture features in the FPGA. The host is a PC-Pentium 4, and works as a Control Unit (CU), which loads different input images for each stage of the external memory of the FPGA. The input image is divided into sub-regions of size $128 * 128$, and GLCMs are calculated for four distance $d = \{1, 2, 3, 4\}$ and four directions $\theta = 0, 45, 90, 135$ at the same time. In total, they calculate 16 GLCMs. The sub-region with $N * N$ is stored into four SRAM's bank to read four neighbor pixels of reference pixel in parallel. The block diagram of calculation GLCM is shown in Figure 3.5. The process starts with reading the first pixel of each

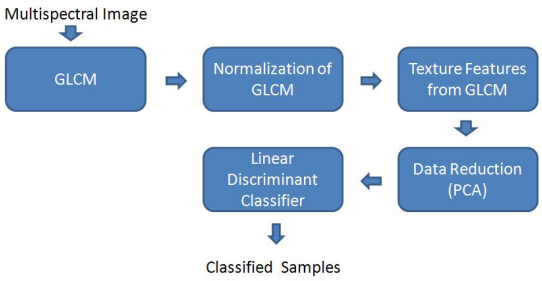

Figure 3.3: Algorithm for the classification of prostate tissue cancer [4].

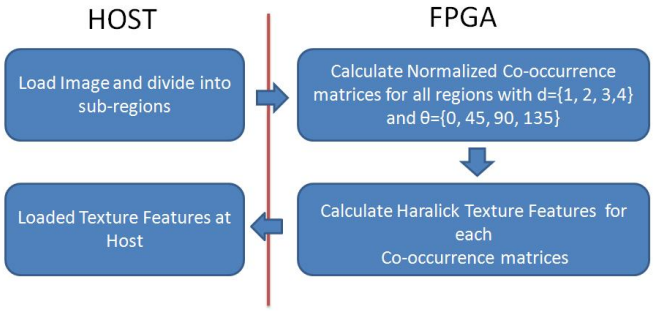

Figure 3.4: System model [4].

bank; these pixels are known as reference pixel, then their four neighbors are read in 4 clock cycles, after that the memory address of all 16 GLCMs are calculated in parallel and the number of occurrences of pixels in co-occurrence matrix is updated, the process is repeated for all of the image pixels [42]. After calculated all 16 GLCMs in parallel, these matrixes are normalized in parallel too. The final results of normalization are stored into SRAM of the FPGA. Then the calculation and normalization are repeated for other sub-regions. The rest of their research for calculation texture features are the same with their pervious jobs [29]. Their results show that the performance of FPGA is 5 times faster than the Pentium 4PC, the reason is related to calculate GLCMs in parallel, and the computation time is independently of size of the input image. Each input image is divided into sub-regions with size $N * N$, that the best value for the N is 128. FPGA executes one sub-region at the time, and others are in the loop.

Iakovidis et al. [5](2004) presented an FPGA based architecture for real time image feature extraction using GLCM analysis in 2004. They implemented their hardware module on Xilinx Virtex-E V2000 FPGA. Their design calculates GLCMs and GLCM integer features in parallel, and their architecture is combination of hardware and software to raster scan input images with sliding windows and calculate 16 dimensional feature vectors consisting of 4 GLCM features for 4 directions. They calculate four of texture features, namely, angular second moment, correlations, inverse difference moment, and entropy. Their architecture has two steps, a preprocessing stage and the feature extraction block. Figure 3.6 is an overview of the architecture.

In the preprocessing steps, the input image is prepared to be processed by the feature

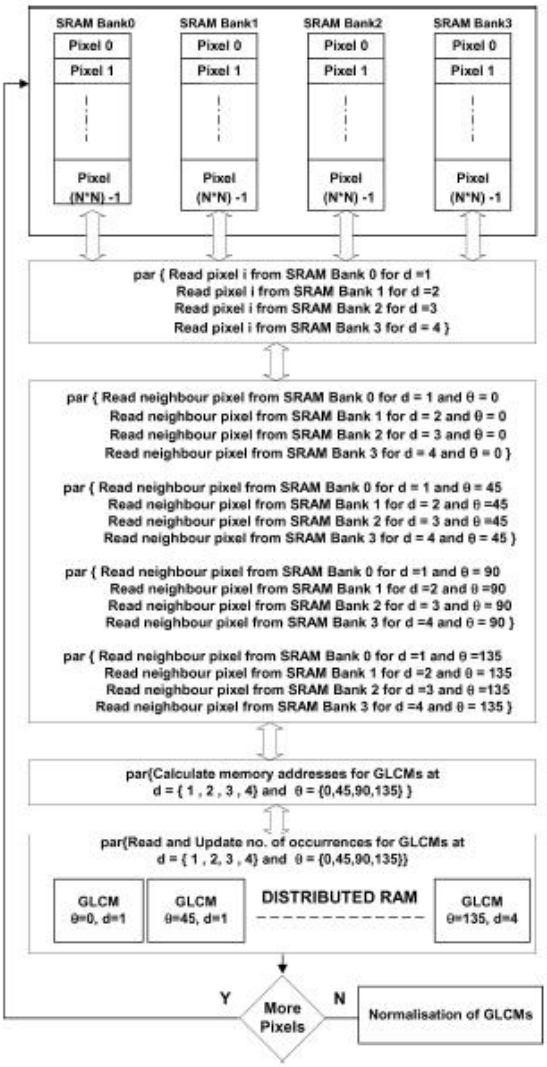

Figure 3.5: Block diagram of calculation GLCM on FPGA [4].

extraction block. For preparation, the input image is convert to an array A, each element of the array is presented by $\overline{a} = \{a_0 a_1 a_2 a_3 a_4\}$, (5 integers) that is related to each pixel, a_0 is the gray-value of the reference and other are the gray-values of reference pixel in four directions. They assumed, the gray-value is up to 64 that can be presented by 6 bits so each element is shown by 30 bits and read in 1 clk. The second step is a combination of hardware and software for calculation GLCM features, the feature extraction block includes hardware and software module. The hardware module is implemented on a Xilinx Virtex-E V2000 FPGA and the FPGA is hosted by the Celoxica RC1000 card. The host preprocesses the input image and presents each pixel as a one element of the array A and loads the result into one of the four memory banks on the card, the FPGA calculates the feature vectors and stores them into in another memory bank [5].

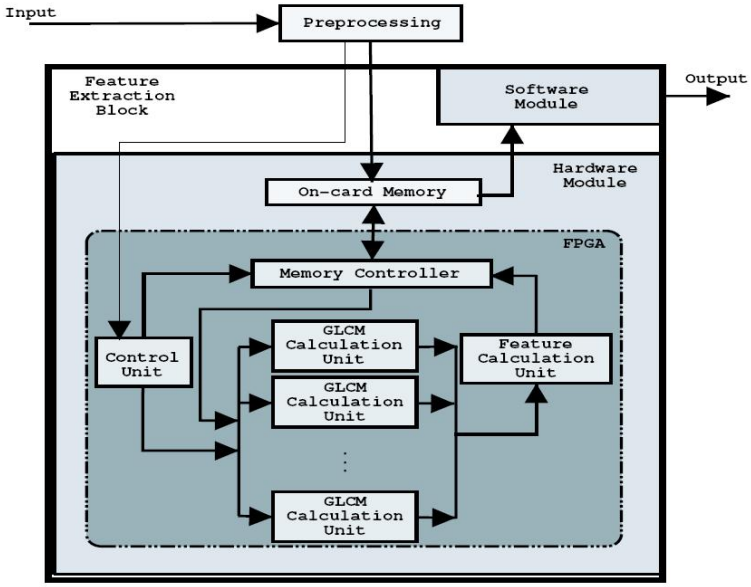

Figure 3.6: Overview of the FPGA architecture [5].

The FPGA architecture consists of Control Unit (CU), Memory Controller, GLCM Calculation Unit (GLCMCU), and Feature Calculation Unit. The CU generates signals that synchronize the other units to coordinate the FPGA's functionality. The memory controller handles the transactions from and to the on-card memory. The GLCMCU receives pairs of gray-value of reference pixel and one of its neighbors as input. The input of feature extraction unit is a GLCM generated by each GLCMU and its output, that is a vector $\overline{V} = \{V_1, V_2, V_3, V_4, V_s\}$ is stored on the on-card memory. As mentioned before, they consider to four of texture features, they replaced the integer operations instead of floating point operations to simplify the hardware implementation [5]:

1. Angular Second Moment (ASM)

$$f_1 = (\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij}^2)/r^2. \tag{3.1}$$

2. Correlation (COR)

$$f_2 = (r.N_g^2 \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} i.j.c_(ij) - r^2).\frac{N_g - 1}{S} \tag{3.2}$$

$$S = \sum_{k=1}^{N_g}(r - C_x(k))^2 \tag{3.3}$$

3. Inverse Difference Moment (IDM)

$$f_3 = 2^{-30}.\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij}.IDMLUT[i-j] \tag{3.4}$$

4. Entropy (ENT)

$$f_4 = 2^{-26}.\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij}.(LOGLUT[c_{ij}] - 2^26.logr) \tag{3.5}$$

Where $c_{i,j}$ is the ijth 16-bit register
Vectors are defined by these equations:

$$V_1 = \sum c_{ij}^2, \tag{3.6a}$$

$$V_2 = \sum i.j.c_(ij), \tag{3.6b}$$

$$V_3 = \sum c_{ij}.IDMLUT[i-j], \tag{3.6c}$$

$$V_4 = \sum c_{ij}.(logc_{ij} - logr), \tag{3.6d}$$

$$V_S = \sum (r - C_x(k))^2. \tag{3.6e}$$

The software module, reads the vectors V and converts the integer component of each vector into 32-bit floating point values, and calculates the corresponding GLCM features. In this design, they use integer operations and also their input image has a limitation for gray-values up 64 bit.

Iakovidis et al. [3](2006) presented a dedicated hardware system for the extraction of second-order statistical features from high-resolution images, they extracted four of Haralick texture features, ASM, COR, IDM, and ENT. The input images can have a resolution from 512×512 to 2048×2048 pixels. They implemented their architecture on a Xilinx VirtexE-2000 FPGA and used integer arithmetic, a sparse co-occurrence matrix representation and a fast logarithm approximation to improve efficiency. Each image is divided into blocks of user-defined size and a feature vector is extracted for each block. Their system calculates a symmetric GLCM for four directions $0, 45, 90, 135$ and for distances so in total 16 GLCM are calculated, and four feature vectors on the same FPGA core in parallel. There are CU, and CMCU, Vector Calculation Units (VCUs) in their design. For each direction, one CMCU is used. They define N_g as 64 bits instead of 32 bits that used in [4], and changed their hardware. These CMCUs are designed for achieving to the three objects, as follows,

- Small FPGA area utilization that makes just one core is used for calculation of GLCM and four VCUs
- High throughout per clock cycle

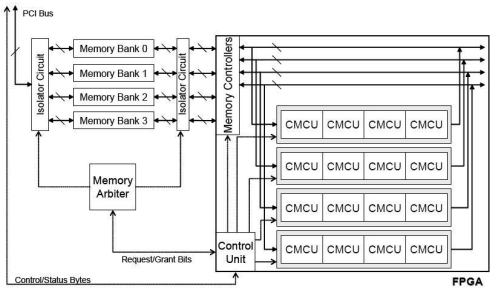

Figure 3.7: Overview of the FPGA architecture [4].

- High frequency potential

They used a $\overline{v} = \{v_1, v_2, v_3, v_4, v_5\}$ for calculating features. This design is suitable for high regulation video, analysis of the multiple video streams. Furthermore, by using one core, there is no overhead by reprogramming cache core onto the FPGA. Each pixel is represented by 25 bits so at each CLK 25 bits from each memory bank is read when in Tahir et al design each pixel is represented by 5 bits. By using set-associative arrays for each sparse GLCM, four vectors can be calculated in a single core. They just implemented four features of Haralick [3].

Iakovidis et al. [4](2007) present a FPGA architecture for fast parallel computation architecture of co-occurrence matrices in high throughput image analysis applications that performance is an important, and they extracted four of Haralick texture features, namely, ASM, COR, IDM, and ENT. Their target device is Xilinx Virtex XCV2000E-6 FPGA. This architecture calculates a symmetric and sparseness co-occurrence matrices in four directions $0, 45, 90, 135$ and for distances (16 GLCMs), and also four feature vectors on the same FPGA core in parallel. The input The input image is divided into blocks of user-defined size and a feature vector is extracted for each block, each pixel of the image is presented by a vector $\overline{a} = \{a_p, a_0, a_{45}, a_{90}, a_{135}\}$, that a_p is the grey-level of the reference pixel and others are the grey-levels of its neighboring. The architecture of FPGA for GLCM shows in Figure 3.7. This architecture consists of a control unit, sixteen Co-occurrence Matrix Computation Units (CMCUs), and n memory controllers that each of them is for one memory bank. In parallel up to n input images of N_g grey-levels can be loaded in memory banks. They use one FPGA core for calculating GLCM and features so there in no overhead by reprogramming each core onto FPGA [4]. The Control Unit (CU) organizes all FPGA functions, creates synchronization signals for the memory controllers and the CMCUs, and communicates with the host, by exchanging control and status bytes, and request or release the rights of the memory banks. Each CMCU is used for calculating the co-occurrence matrix of an image for a different direction and distance [4]. Their results show the feasibility of real-time feature extraction for input images of dimensions up to $2048x2048$ pixels, where a performance of 32 images per second is achieved. This architecture calculates a symmetric and sparseness of the GLCMs to achieve improved processing times, and smaller, flexible area utilization

4 FPGA Implementation

In this chapter, the proposed model for calculating GLCM is presented. The chapter is organized as follows. The first section represent a description of FPGA, and comparison between FPGA and ASIC. Section Section 4.1.1 presents the description of hardware targets. The function of design is described in section Section 4.2. The design was developed in Very High Speed Integrated Circuits Hardware Description Language (VHDL), and three hardware targets are used for calculation of 16 GLCMs in four distance with four angles. The common value for angles are $0^o, 45^o, 90^o, 135^o$ [29, 4, 42], and the values of distances are $d = 1, 2, 3, 4$. In last section, the results and their analysis are presented.

4.1 FPGA

Field Programmable Gate Array (FPGA) is a semiconductor device that can be programmed by user after manufacturing and implemented by any logical functions that an Application Specific Integrated Circuit (ASIC) could perform, these abilities propose advantages for many applications. FPGAs consist of various mixes of embedded SRAM, high-speed I/O, logic blocks, and routing. In particular, an FPGA has a programmable logic components, which called logic blocks and a hierarchy of reconfigurable interconnects. Logic blocks consist of a Look-Up Table (LUT) for logical functions and memory elements or blocks of memories, which may be simple flip-flop or more complete blocks of memory for storage [21]. Reconfigurable interconnects allow the logic blocks to be wired together.

4.1.1 FPGA vs ASIC

As opposed to ASICs, FPGAs can be programmed in several times based on design and memory bits and logic gates. However, ASICs have high development cost and time consuming development procedure and just memory bits are controlled by user. On the other hand, FPGAs are slower than ASIC. The choice of whether to use of FPGA or ASIC is based on design, the chip will need to be reprogrammed or not, and cost. Sometimes, first design is prototyped on FPGA and after find the stable design, is implemented on ASIC. One of the applications that FPGAs are used is real time image processing that needs to be run in parallel. Image and video application need a wide area but based on limitation in memory bandwidth, and confliction in resource (e.g. local and off-chip RAM), the input image is divided into equal parts and processing is done in multiple pipeline process.

4.2 Hardware Target

The hardware targets are Xilinx Virtex-XC2VP30, Virtex- XC4VfX60, and XC5VLX330.

Virtex-XC2VP30 is characterized by 3424 CLBs providing 13,696 slices, which each CLB is 4 slices, and has 136 18 Kb True Dual-Port Bock RAM. Table 4.1 shows the available resources in virtex-XC2VP30. Each logic cell has one 4-input LUT, one Flip Flop (FF), and carry logic.

Device	RocketIO transceiver Blocks	PowerPC Processor Blocks	Logic Cells	CLB (1 = 4 slices = max 128 bits)		18 × 18 Bit Multipler Blocks	Block SelectRAM+		DCMs	Maximum User I/O Pads
				Slices	Max Distr RAM(kb)		18 Kb Blocks	Max Block RAM(kb)		
XC2VP30	8	2	30816	13696	428	136	136	2448	8	644

Table 4.1: Virtex-XC2VP30.

Virtex-XC4VfX60 conta 6320 CLBs providing 25,280 slices, which each CLB is 4 slices. Table 4.2 depicts the available resources in this virtex. Each CLB is 4 slices and optimized by 64 bits, is means that at each cycle, 64 can be access.

Device	Configurable Logic Blocks (CLBs)				XtremeDSP Slices	Block RAM		DCMs	PMCDs	PowerPC Processor Blocks	Ethernet MACs	Total I/O Banks	Max User I/O
	Array Row ×Col	Logic Cells	Slices	Max Distributed RAM (Kb)		18Kb Blocks	Max Block RAM (Kb)						
XC4VfX60	128×52	56880	25280	395	128	232	41760	12	8	2	4	13	576

Table 4.2: Virtex-XC4VfX60.

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices	Block RAM Blocks			CMTs	PowerPC Processor Blocks	Endpoint Block for PCI Express	Ethernet MACs	Max RocketIO Transeiver		Total I/O Banks	Max User I/O
	Array Row ×Col	Virtex-5 Slices	Max Distributed RAM (Kb)		18 (Kb)	36 (Kb)	Max (Kb)					GTP	GTX		
XC5VLX330	240×108	51.840	3420	192	576	288	10368	6	N/A	N/A	N/A	N/A	N/A	33	1200

Table 4.3: Virtex-XC5VLX330.

Virtex-XC5VLX330 has a 6-input LUT, and 36Kb BRAM. The 4-input LUT has a truth table capacity for 16 different combinations but with 6-input LUT, the truth table is increased to 64 different combinations. Table 4.3 depicts the available resources in Virtex- XC5VLX330. Comparison with other Virtex, it has more memory, low power and higher speed than virtex4.

All three Virtexs have a true dual port BRAMs. Each port has an individual clock, enable, reset, write enable, address, data input, and data output lines. The advantage of dual port is that we can access to different address in BRAM at the same time for read and write operation.

4.3 Proposed Design

There are different ways to calculate GLCMs. In the simplest way, GLCMs are serially calculated. This way takes more time because each pixel should be compared with each

of its neighborhood pixels one by one. Another way is calculation 4 GLCMs in parallel for four directions and one distance. In the third way, which has been implemented in this thesis, GLCMs are calculated for four distances $d = 1, 2, 3, 4$ and four directions $0^o, 45^o, 90^o, 135^o$ in parallel. This technique increases the throughput. These different distances and directions are depicted in Figure 4.1

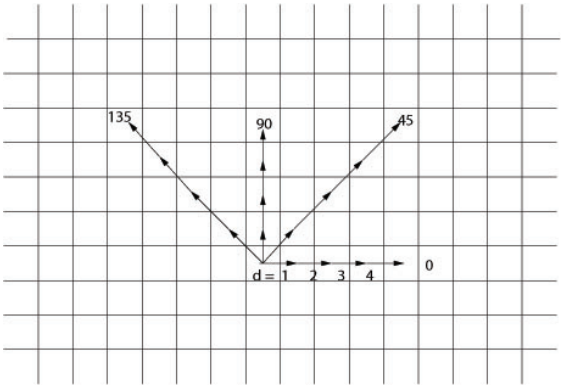

Figure 4.1: Different distances with four different directions, which have been used to calculate sixteen GLCMs.

The proposed design consists of three main parts. First, an image is divided into sub-regions. Second, each sub-region is loaded. Finally, sixteen GLCMs are computed for loaded sub-region. The last two steps are repeated for all sub-regions. These stages are depicted in Figure 4.2.

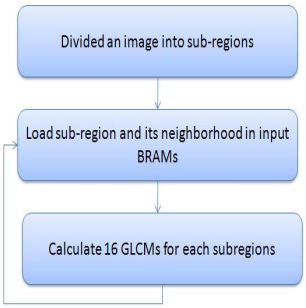

Figure 4.2: Proposed model.

The image sizes of 32×32, 64×64, and 128×128 have been used in our implementations. In our design, we used BRAMs for storing input image and calculated GLCMs. The number of BRAMs depend on the image size and the number of gray levels (N_g). We generated these BRAMs with core generator. These BRAMs are dual port. It is possible that those BRAMs are defined either one port or dual ports. We defined two types of BRAMs, input BRAMs and output BRAMs. The input BRAMs are defined as a single port and input image is stored into them. If the size of the input image is small then that image can be stored in a BRAM, while if the image is large it can be stored in several BRAMs. Four BRAMs are also used for storing the neighboring

pixels at four distances and four directions. The output BRAMs are defined as dual port. The calculated GLCMs are stored in output BRAMs. The reason to define these BRAMs as a dual port is that the calculated GLCMs are symmetrical. Hence, we need to write same value into different address at the same time. For example, for image size 32×32 with $N_g = 32$, there are 25 BRAMs, one BRAM for storing the original image (sub-regions) that each pixel of the original image is known as a reference pixel, eight BRAMs that each of two BRAMs for storing neighboring pixels of each reference pixel at four directions $0^o, 45^o, 90^o, 135^o$ with four distances $d = 1, 2, 3$, and 4, and 16 BRAMs is used to store 16 calculated GLCMs. The overview of the FPGA architecture with these assumptions is depicted in Figure 4.3.

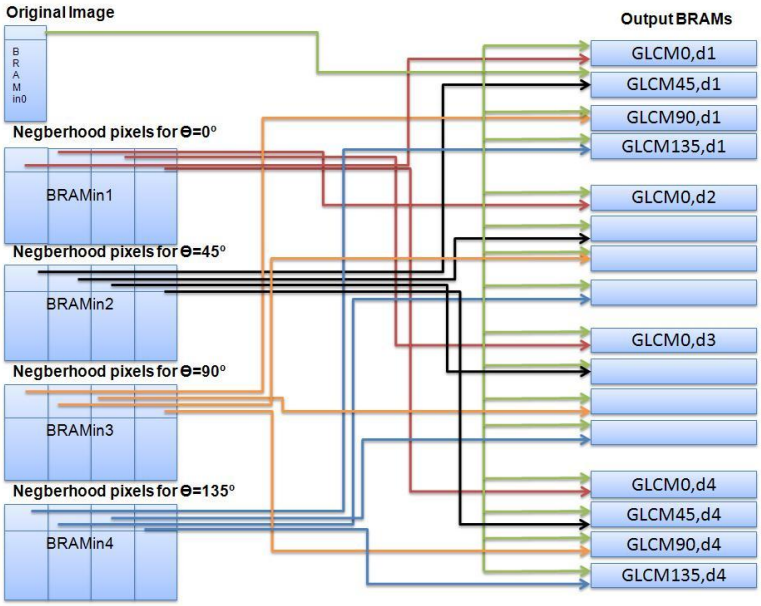

Figure 4.3: Overview of the input and output BRAMs for the image size 32×32 and $N_g = 32$.

As this figure shows, five BRAMs are used as input BRAMs and sixteen BRAMs are used as output BRAMs. The first input BRAM has $N \times N$ M-bit, where $N \times N$ is the size of the image and M is the number of bits to represent each pixel of image. In other words, $N_g = 2^M$. Each BRAM of other input BRAMs has K $4 \times M$-bit, where 2^K is $N \times N$. This is because we have considered four neighboring pixels at each distance for each reference pixel. The size of each output BRAM is $2^M \times K$-bit. Figure 4.4 depicts the architecture of input BRAMs and output BRAMs.

Our design contains 17 Processing Unit (PU), one PU is used for reading a pixel from the input image, and other PU read its neighborhood and calculate GLCMs or load new value on output BRAMs. All these PU are run in parallel Figure 4.5 shows the architecture of our design for reading a reference pixel and one of its neighboring from input BRAMS and loading data on output BRAM. These PUs work in parallel. In the first PU, a reference pixel is read from input BRAM0, each other PU read one of the sixteen neighborhood of the reference pixel. For example, the second PU reads

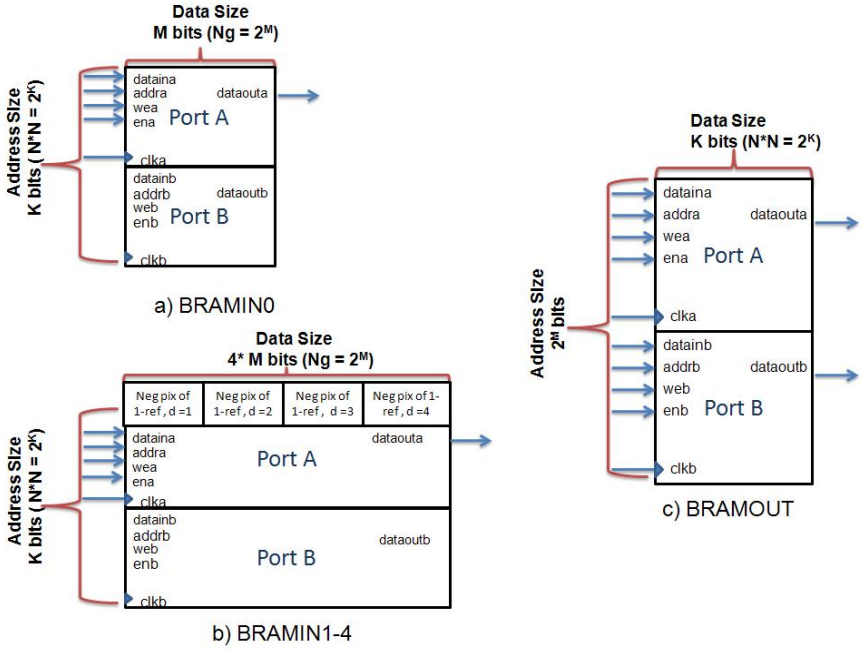

Figure 4.4: Architectures of BRAMs for $N_g = 32$.

the pixel with $d = 1$ in directions 0^o that located into last M bits of input BRAM1, and PU6 reads the pixel with $d = 2$ in directions 0^o that is located into the third M bits of BRAM1. With the value of gray tone of these pixels address for accessing to output BRAMs are calculated, and data in this address are increased by one. As mentioned in pervious chapter there are two types of GLCM, symmetrical, and non-symmetrical. In this design, symmetrical GLCMs are calculated by generating output BRAMs as true dual port, so at the same time, the same value can be write into two different addresses. For example, if the gray tone value of reference pixel is known as a index i, and the gray tone value of one of the neighborhood pixel is known as an index j, at the same time the value in (i, j) and (j, i) will be increased.

4.4 Results

Experimental evaluation has been performed based on performance and area utilization. The proposed technique has been simulated on three target platforms, Virtex-XC2VP30, Virtex-XC4VfX60, and Virtex-XC5VLX330. The image sizes are 32×32, 64×64, and 128×128, with three $N_g = 32, 64, 128$ values.

The Processing Time (PT) can be calculated either by simulations or by theoretically. The PT is the multiplication of total execution cycles with the cycle period, when the simulations are used. while, from the theoritically point of view, the PT is equal to the multiplications of clock period with the number of image pixels and the number of cycles that needed to process each image pixel (Np). PT = clock period x N x N x Np. where $N \times N$ is the dimension of an image.

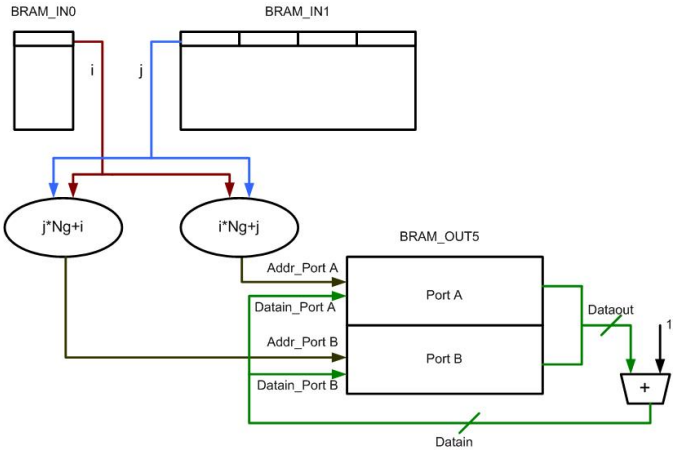

Figure 4.5: Processing units.

The clock frequency is based on how a hardware design can be optimized. First, we expected that clock period of the Virtex5 should be lower than others but it was not, it appeared that Virtex2 has the lowest clock period. The reason is that, we have a critical path for accessing to the Block RAMs. In different Virtex devices, this time is different. In general, the accessing time is related to the hardware design and the size of Block RAM. With increasing a Block RAM size, the memory access times will be increased. In Virtex5, BRAM size is 36Kb, hence it takes longer time to read data from BRAMs than other Virtex devices. Figure 4.6, Figure 4.7, and Figure 4.8 depict the obtained clock period and frequency on different Virtex platform for different image size.

Ng = 32			
Size of Image	32*32	64*64	128*128
XC5VLX330-2ff1760			
Frequency (MHz)	254.514	256.276	250.853
Clock period (ns)	3.929	3.902	3.986
XC4VfX60-12ff1152			
Frequency (MHz)	271.172	233.68	230.792
Clock period (ns)	3.688	4.279	4.333
XC2VP30-7ff896			
Frequency (MHz)	357.481	359.861	352.038
Clock period (ns)	2.797	2.779	2.841

Figure 4.6: Frequency and clock period $N_g = 32$.

Based on the results for clock period, we computed processing times theoretically for our experiments. The achieved processing times for computation 16 GLCMs are depicted in Figure 4.9.

Based on our achievements, we can conclude, the variance in processing time is related to different frequencies of the different FPGA devices, and the design.

Figure 4.10 until Figure 4.12 show comparison graph of the processing time for different image size with the same gray-tone values in different device.

Ng = 64			
Size of Image	32*32	64*64	128*128
XC5VLX330-2ff1760			
Frequency (MHz)	254.51	256.276	250.853
Clock period (ns)	3.929	3.902	3.986
XC4VfX60-12ff1152			
Frequency (MHz)	271.172	233.68	230.792
Clock period (ns)	3.688	4.279	4.333
XC2VP30-7ff896			
Frequency (MHz)	378.903	361.253	
Clock period (ns)	2.639	2.768	

Figure 4.7: Frequency and clock period $N_g = 64$.

Ng = 128		
Size of Image	32*32	64*64
XC5VLX330-2ff1760		
Frequency (MHz)	206.286	207.441
Clock period (ns)	4.848	4.821
XC4VfX60-12ff1152		
Frequency (MHz)	206.409	185.775
Clock period (ns)	4.845	5.383

Figure 4.8: Frequency and clock period $N_g = 128$.

Based on the above figures, The XC2VP30 has the lowest processing time in our design. However, the number of BRAM available in XC2VP30 is less than other platforms, and it cannot supported an input image with the size 128×128 and also for image with $N_g = 128$. To deal with this limitation, we can make exception for the input image, and the data size of output BRAMs. In general, the output BRAMs data size is the maximum number of pairs (i, j) that is 2^N, where N is image size. By make limitation for the number of occurrence, we can have 128×128 with $N_g = 64$ or $N_g = 128$ as an input image for XC2VP30. For example, when the image size is 128×128, with $N_g = 128$, the maximum number is 16384 that represents by 14-bits. In that case, for using XC2VP30, we assumed the maximum number of occurrence is 2^9, hence data size of output BRAMs is 9 bits. Figure 4.13 shows the new result with this limitation for all Virtex devices.

Another conclusion is that, Vritex5-330 has a better throughput than XC4VFX60 for a bigger image. The number of resources that used in Virtex5 is less than Virtex4 and the processing time is better.

The area utilization of different implementations are depicted from Figures 4.14 to Figure 4.23.

We can make some conclusions regarding the results of the area utilization. One of the conclusion is that, implementation on Virtex-XC5VLX330 uses less resources compared to the other Virtex devices, except the number of used slices. The size of BRAMs in

Ng	32			64			128		
Size of image	32*32	64*64	128*128	32*32	64*64	128*128	32*32	64*64	128*128
XC5VLX330-2ff1760	20.1	79.9	326.5	20.1	79.9	326.5	24.8	98.7	
XC4VfX60-12ff1152	18.9	87.6	355	18.9	87.6	355	24.8	110.2	
XC2VP30-7ff896	14.3	56.9	232.7	13.5	56.7				

Figure 4.9: Processing times(μs)for different image dimensions and various N_g by using various FPGA devices.

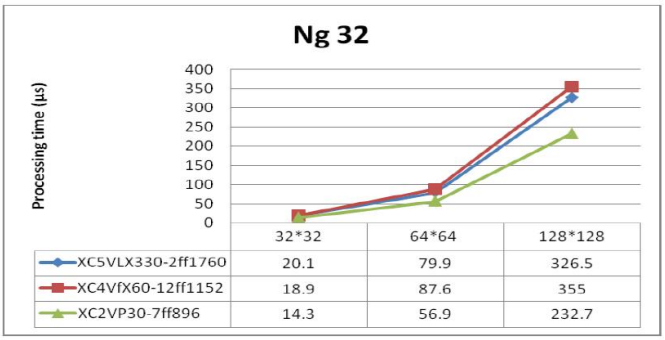

Figure 4.10: Comparison of processing times (μs) with $N_g = 32$.

Virtx-XC5VLX330 is bigger than the others. Hence, less number of BRAMs are used. In addition, Virtex-XC5VLX330 has a 6-input LUT, while the other Virtex devices have 4-input LUT. Therefore, less number of LUT will be used in virtex5. Virtex4 and Virtex2 use the same number of BRAMs, because their BRAMs have the same size. However, virtex4 uses more number of slices and number of slices flip-flops. Another conclusion is with increasing the image size, the number of slices used will be increased too except in XC4VFX60 when the size of image is 64×64. The reason of this exception is that, Virtex4 is optimized by 64 bits, and one CLB is 64 bits. Hence, when image size 64, number of resource used will be optimized. Furthermore, the memory complexity is $O(2^{2K} \times M)$, K can be found from this equation $N^2 = 2^K$, where N is image size, and M is the number of bits that each pixel is represented.

4.5 Comparison with Related Work

In order to show the effectiveness of our implemented design, we have compared our result with one recent related work in [4]. They proposed FPGA architecture for fast parallel computation of co-occurrence matrices. They computed sixteen GLCMs for the image size between 16×16 up to 512×512 with $N_g = 32$. Table 4.4 lists their result.

Our architecture also computes sixteen GLCMs in parallel. The difference between our design and their design is that, we used Block RAM to store image and GLCMs, and they stored the input image in four 2MB external RAM. Based on the limitation of the number of available BRAMs, we have limitation for image size, hence we cannot

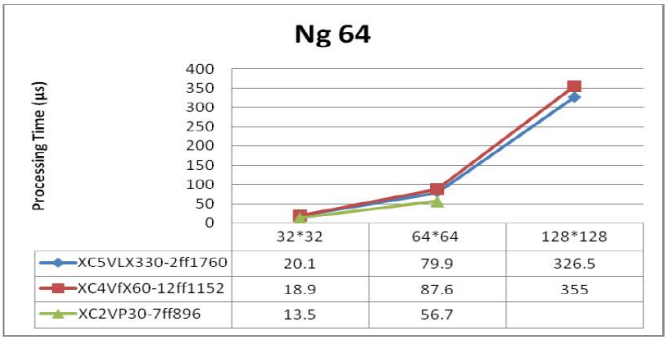

Figure 4.11: Comparison of processing times (μs) with $N_g = 64$.

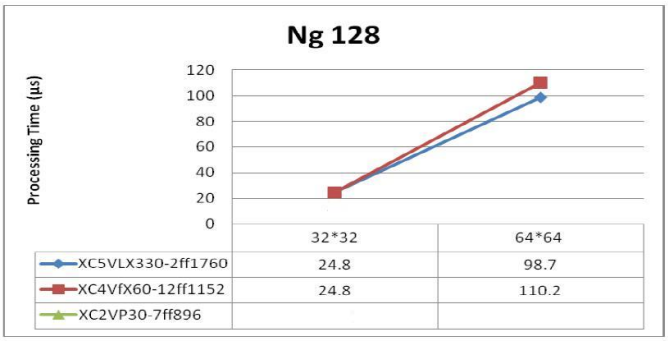

Figure 4.12: Comparison of processing times (μs) with $N_g = 128$.

use image with size more that 128×128. Table 4.5, Table 4.6, and Table 4.7 show comparisons of our results with their best results for three different image size.

Our results show that depending on the in-use Virtex, our implemented algorithm can improve the computation time of GLCM two to four times compared to the implemented algorithm in Iakovidis et al [4]. With this speed in calculation GLCMs, features can be calculated 2x up 4x faster compared to [4].

Ng	64
Size of Image	128*128
XC5VLX330-2ff1760	330
XC4VfX60-12ff1152	344.5
XC2VP30-7ff896	71.1

Figure 4.13: Processing times (μs) with limitation of number of occurs for 128×128 image.

Ng = 32					
	Number of Slices Registers	Number of Slice LUTs	Number of fully used LUT-FF pairs	Number of Block RAM/FIFO	Size of image
XC5VLX330-2ff1760	466 (0%)	430 (0%)	230 (34%)	13 (4%)	32*32
	500	336	264 (46%)	19 (6%)	64*64
	546 (0%)	690 (0%)	298 (30%)	47 (16%)	128*128
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs	Number of FIFO16/RAMb16s	Size of image
XC4VfX60-12ff1152	336(1%)	446(0%)	462(0%)	25(10%)	32*32
	319	500	384	38 (16%)	64*64
	500 (1%)	546 (1%)	754 (1%)	93 (40%)	128*128
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs	Number of BRAMs	Size of image
XC2VP30-7ff896	266 (1%)	466 (1%)	350 (1%)	25 (18%)	32*32
	307 (2%)	507 (1%)	352 (1%)	38 (27%)	64*64
	490 (3%)	556 (2%)	722 (2%)	93 (68%)	128*128

Figure 4.14: Area utilizations $N_g = 32$.

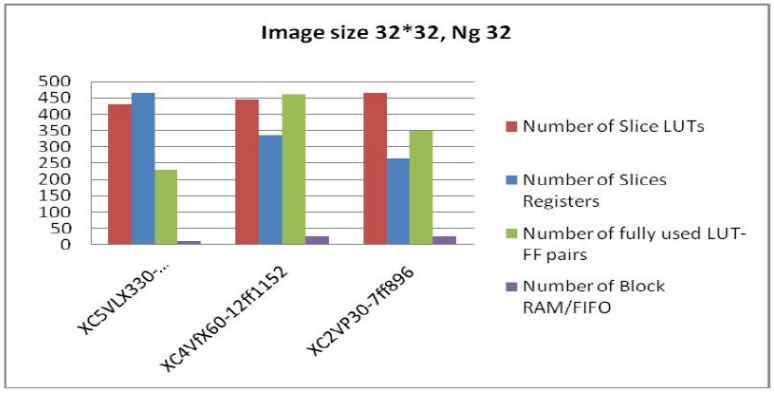

Figure 4.15: a) Area utilizations.

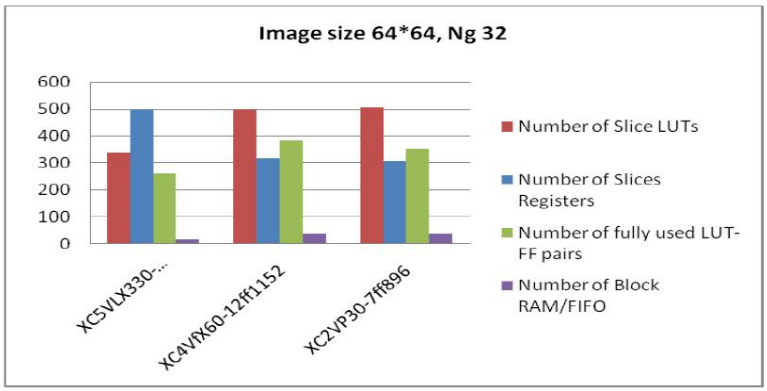

Figure 4.16: b) Area utilizations.

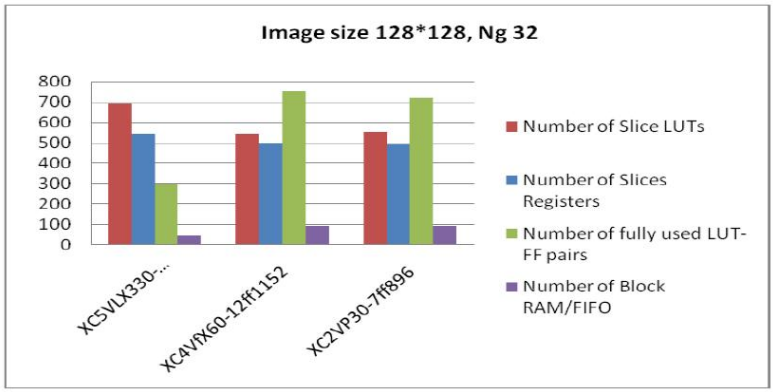

Figure 4.17: c) Area utilizations.

Ng = 64					
	Number of Slices Registers	Number of Slice LUTs	Number of fully used LUT-FF pairs	Number of Block RAM/FIFO	Size of image
XC5VLX330-2ff1760	515 (0%)	430 (0%)	230 (34%)	29	32*32
	549 (0%)	336 (0%)	264 (46%)	37 (12%)	64*64
	595 (0%)	690 (0%)	298 (30%)	79 (27%)	128*128
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs	Number of FIFO16/RAMb16s	Size of image
XC4VfX60-12ff1152	364 (1%)	515 (1%)	462 (0%)	57 (24%)	32*32
	347 (0%)	549 (0%)	384	74 (31%)	64*64
	528 (2%)	595 (1%)	754 (1%)	158 (68%)	128*128
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs	Number of BRAMs	Size of image
XC2VP30-7ff896	294 (2%)	515 (1%)	350 (1%)	57 (41%)	32*32
	336 (2%)	557 (2%)	352 (1%)	74 (54%)	64*64

Figure 4.18: Area utilizations $N_g = 64$.

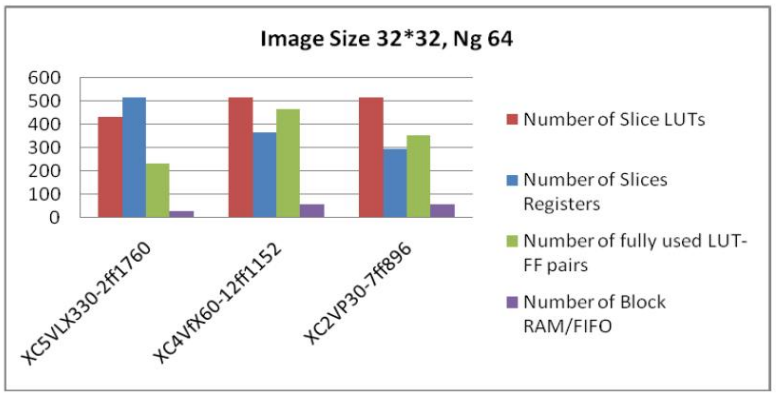

Figure 4.19: a) Area utilizations.

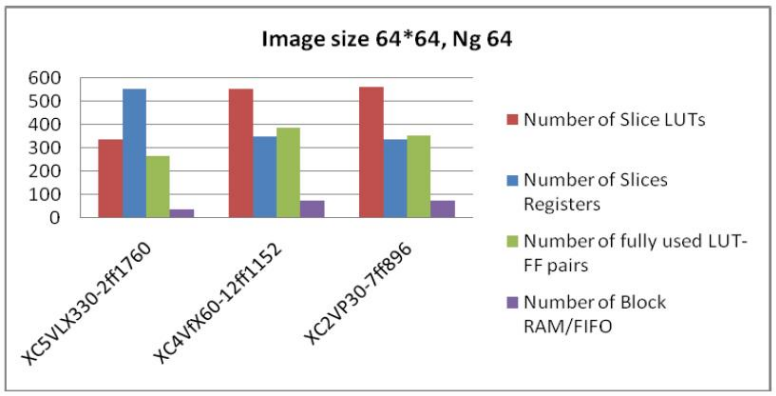

Figure 4.20: b) Area utilizations.

Ng = 128					
	Number of Slices Registers	Number of Slice LUTs	Number of fully used LUT-FF pairs	Number of Block RAM/FIFO	Size of image
XC5VLX330-2ff1760	660 (0%)	1838 (0%)	244 (10%)	77 (26%)	32*32
	694	1792	264 (11%)	103 (35%)	64*64
	Number of Slices	Number of Slice Flip Flops	Number of 4 input LUTs	Number of FIFO16/RAMb16s	
XC4VfX60-12ff1152	1137 (4%)	660 (1%)	1870 (3%)	153 (65%)	32*32
	1155	694	1840	206 (88%)	64*64

Figure 4.21: Area utilizations $N_g = 128$.

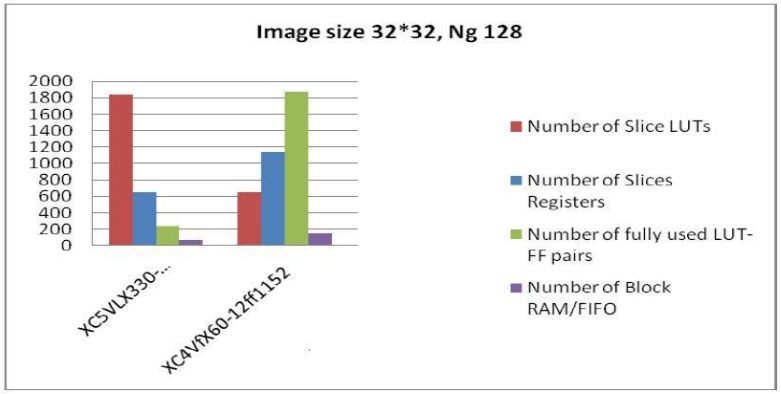

Figure 4.22: a) Area utilizations.

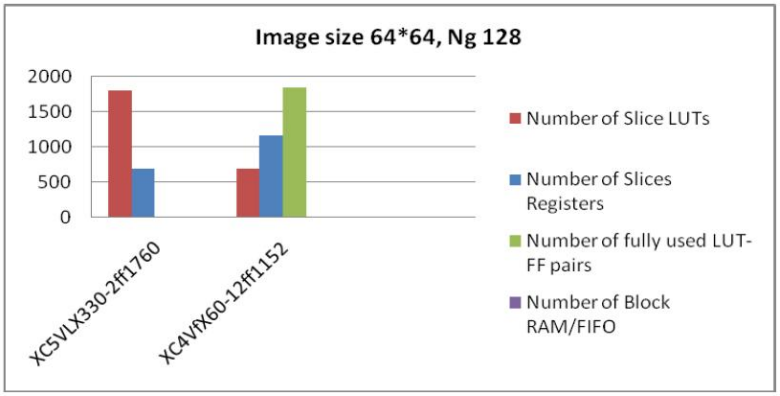

Figure 4.23: b) Area utilization.

N_g		*Image size*					
	Frequency (MHz)	16 x 16	32 x 32	64 x 64	128 x 128	256 x 256	512 x 512
XCV2000E-6	38						
Processing times (μs)		30	113	442	1,756	7,013	28,041
XCV2000E-8	51						
Processing times (μs)		22	93	323	1,283	5,123	20,483
XC3S4000E-5	72						
Processing times (μs)		15	59	230	915	3,653	14,606
XC2V6000E-6	83						
Processing times (μs)		13	51	198	788	3,149	12,590

Table 4.4: Processing times (μs) achieved for various input image dimensions using various FPGA devices. Iakovidis et al [4].

$Image\ size 32 \times 32$	
$N_g = 32$	$Processing times(\mu s)$
$XC5VLX330 - 2ff1760$	20.1
$XC4VFX60 - 12ff1152$	18.9
$XC2VP30 - 7ff896$	14.3
$XC2V6000E - 6$ (Iakovidis et al)	51

Table 4.5: a) Comparison the processing time (μs) our result with the Iakovidis et al results

$Image\ size 64 \times 64$	
$N_g = 32$	$Processing times(\mu s)$
$XC5VLX330 - 2ff1760$	79.9
$XC4VFX60 - 12ff1152$	87.6
$XC2VP30 - 7ff896$	56.9
$XC2V6000E - 6$ (Iakovidis et al)	198

Table 4.6: b) Comparison the processing time (μs) our result with the Iakovidis et al results

$Image\ size 128 \times 128$	
$N_g = 32$	$Processing times(\mu s)$
$XC5VLX330 - 2ff1760$	326.5
$XC4VFX60 - 12ff1152$	355
$XC2VP30 - 7ff896$	232.7
$XC2V6000E - 6$ (Iakovidis et al)	788

Table 4.7: c) Comparison the processing time (μs) our result with the Iakovidis et al results

5 Conclusion and Future Work

This chapter presents the conclusions from this thesis. In section, we provide a summary of the thesis. Future work are proposed in section .

5.1 Conclusions

Digital image has several features, such as, texture, color, shape, and etc. Texture is one of the important features. There is no universally agreed definition for texture. It can be described as the structural information pattern of an image. Texture analysis has an important role in image processing, computer vision and pattern recognition. Texture extraction is the first step of texture analysis. In this step, the basic information from image is extracted. Texture features or properties have useful information about an image. There are four main methods to extract texture features, namely, statistical, structural, model-based, and transform. We have chosen the Gray Level Co-occurrence Matrix (GLCM) method to extract texture features, which is a statistical method based on the gray level value of pixels. This method was proposed by Haralick [16] on (1979). Each element of the GLCM, $p(i, j)$, is a number of occurrence that two pixels with gray levels i and j at a distance d in a given directions are neighborhood. Generally, GLCM can be a symmetrical or non-symmetrical matrix. Several statistical features can be extracted from GLCM, which represents texture properties, such as mean, contrast, and variance. However, the calculation of GLCM is computationally intensive.

However, Calculation of Gray Level Co-occurrence Matrix (GLCM), which is used to extract Haralick texture features [17] is an example. For an image of size 5000×5000, time required is approximately 350 seconds using Pentium 4 machine. The calculation of the GLCM and Haralick texture features take 75% and 19% of the total time, respectively [42].

In this thesis, we proposed an FPGA-based architecture for parallel computation of symmetric GLCM. Symmetrical algorithms are faster than non-symmetrical and also a hardware implementation consumes less area and less power compared to a software implementation. We presented an FPGA architecture which is capable to calculate GLCMs in parallel for four different distances in four directions. Experiments are done using three different Virtex devices, three image sizes (128×128, 64×64 and 32×32), and three values for gray tone (32, 64 and 128). We stored the processed image and the generated GLCMs in BRAMs. Our results showed that depending on the in-use virtex, our implemented algorithm can improve the computation time of GLCM two to four times compared to the implemented algorithm in [4]. Further, we found that Virtex-XCV2P30 has a better throughput than Virtex4 and Virtex5.

To summarize, the following items have been done in this thesis:

- Studying different methods for extraction the texture features and selecting the the most common byb appreciate method, i.e., GLCM,
- Implementing and evaluating GLCM-based texture feature extraction method with *C* language and computing texture features of sample images,
- Proposing an FPGA-based hardware architecture for texture extraction and implementing it on 3 different Virtex devices,
- Computing sixteen GLCMs in parallel for three different gray tone values, while the similar related works computes only one gray-tone at each time.
- Comparing the performance of our design with the similar implementations, which are up to 4x slower than our implementation under similar conditions,

5.2 Future Work

In this thesis, we proposed a design able of computing sixteen GLCMs for an image with size up to 128×128, and the maximum value for gray-tone is 128 for image size less than 128×128. This work can be extended for computation GLCMs of bigger image sizes and more gray-tone value. For supporting bigger image, an image can be stored into external RAM, and based on the the external RAM, at each time, 64-bits or 32-bits can be read. Another research direction is on the combination of software and hardware design. The software can read an image from external RAM, or block RAM. Based on data bus of RAM, at each cycle 32 or 64 bits can be read and send to the hardware unit. In the hardware unit, GLCMs can be computed and the results send back to the software part for future operation. In such design, PowerPC of Virtex is programmed by software (C code), and its task is to make connection between external RAM or Block RAM and hardware unit or CCU. This design can be implemented using the Molen reconfigurable architecture. At the current stage, the GLCM texture feature computation is implemented in C. We did wrote a VHDL code for this function but the limited time scope of this project did not allow us to integrate it in our design. A logical step would be integration texture features to our hardware design.

Bibliography

[1] M. Ba, D. Degrugillier, and C. Berrou, *Digital VLSI Using Parallel Architecture for CO-occurrence Matrix Determination*, Int. Conf on Acoustics, Speech and Signal Processing, vol. 4, 1989, pp. 2556–2559.

[2] A. Baraldi and F. Parmiggiani, *An Investigation of the Textural Characteristics Associated with Gray Level Co-occurrence Matrix Statistical Parameters*, IEEE Transactions on Geoscience and Remote Sensing **33** (1995), no. 2, 293–304.

[3] D. Bariamis, D. K. Iakovidis, and D. E. Maroulis, *Dedicated Hardware for Real-Time Computation of Second-Order Statistical Features for High Resolution Images*, International Conference on Advanced Concepts for Intelligent Vision Systems **4179** (2006), 67–77.

[4] D. G. Bariamis, D. K. Iakovidis, and D. E. Maroulis, *FPGA Architecture for Fast Parallel Computation of Co-occurrence Matrices*, Microprocessors and Microsystems **31** (2007), 160–165.

[5] D. G. Bariamis, D. K. Iakovidis, D. E. Maroulis, and S. A. Karkanis, *An FPGA-based Architecture for Real Time Image Feature Extraction*, Proc. of the 17th Int. Conf. on Pattern Recognition, vol. 01, 2004, pp. 801–804.

[6] M. V. Boland, *Quantitative Description and Automated Classification of Cellular Protein Localization Patterns in Fluorescence Microscope Images of Mammalian Cells*, Ph.D. thesis, Univeristy of Pittsburgh, 1999.

[7] S. Brandt, *Use of Shape Features in Content-Based Image Retrieval*, Master's thesis, Helesinki University of Technology, August 1999.

[8] S. Brandt, J. Laaksonen, and E. Oja, *Statistical Shape Features in Content-Based Image Retrieval*, The 15th Int. Conf. on Pattern Recognition **2** (2000), 1062–1065.

[9] D. A. Clausi and M. E. Jernigan, *A Fast Method to Determine Co-occurrence Texture Features Using a Linked List Implementation*, Remote Sensing of Environment **36** (1996), no. 1, 506–509.

[10] D. A. Clausi and Y. Zhao, *Rapid Determination of Co-occurrence Texture Features*, IEEE Int. Geoscience and Remote Sensing Symposium **4** (2001), 1880–1882.

[11] ———, *An Advanced Computational Method to Determine Co-occurrence Probability Texture Features*, IEEE Int, Geoscience and Remote Sensing Symp **4** (2002), 2453–2455.

[12] J. M. H. du Buf, M. Kardan, and M. Spann, *Texture Feature Performance for Image Segmentation*, Pattern Recognition **23** (1990), 291–309.

[13] D. Gabor, *Theory of Communication*, **93** (1946), no. 26, 429–457.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed., Prentice Hall, 2007.

[15] M. Hall-Beyer, *The GLCM Tutorial Home Page*, http://www.fp.ucalgary.ca/mhallbey/tutorial.htm.

[16] R. M. Haralick, *Statistical and Structural Approaches to Texture*, Proceedings of the IEEE **67** (1979), no. 5, 786–804.

[17] R. M. Haralick, K. Shanmugam, and I. Denstien, *"Textural Features for Image Classification"*, IEEE Transactions on Systems, Man and Cybernetics **SMC-3** (1973), no. 6, 610–621.

[18] M. Hauta-Kasari, J. Parkkinen, T. Jaaskelainen, and R. Lenz, *Multi-spectral Texture Segmentation Based on the Spectral Cooccurrence Matrix*, Pattern Analysis and Applications **2** (1999), 275284.

[19] M. H. Horng, X. J. Huang, and J. H. Zhuang, *Texture Feature Coding Method for Texture Analysis and It's Application*, Journal of Optical Engineering **42** (2003), no. 1, 228–238.

[20] http://web.iiit.ac.in/ arul/report/node13.html, *Xst user guide*.

[21] http://www.altera.com.

[22] http://www.celoxica.com.

[23] http://www.cs.auckland.ac.nz/ georgy/research/texture/thesis-html/node7.html.

[24] J. Iivarinen, K. Heikkinen, J. Rauhamaa, P. Vuorimaa, and A. Visa, *A Defect Detection Scheme for Web Surface Inspection*, Int. Journal of Pattern Recognition and Artificial Intelligence **14** (2000), no. 6, 735–755.

[25] M. D. Levine, *Feature Extraction : A Survey*, Proceedings of the IEEE **57** (1969), no. 8.

[26] Y. Liu, X. Zhou, and W. Y. Ma, *Extracting Texture Features from Arbitrary-shaped Regions for Image Retrieval*, IEEE Int. Conf. on Multimedia and Expo (2004).

[27] J. Livarinen, K. Heikkinen, and J. Rauhamaaand P. VuorimaaandA. Visa, *Defect Detection Scheme for Web Surface Inspection*, Int. Journal of Pattern Recognition and Artificial Intelligence, vol. 14, 2000, pp. 735–755.

[28] S. Lopez-Estrada and R. Cumplido, *Decision Tree Based FPGA-Architecture for Texture Sea State Classification*, IEEE Int. Conf. on Reconfigurable Computing and FPGA's **31** (2006).

[29] A. Bouridane M. A. Tahir and F. Kurugollu, *Accelerating the Computation of GLCM and Haralick Texture Features on reconfigurable Hardware*, Int. Conf. on Image Processing **5** (2004), 2857–2860.

[30] B. S. Manjunath and W-Y. Ma, *Texture Features for Image Retrieval*, IBM T.J. Watson Research Center, Image Database: Search and Retrieval of digital Imagery (2002), 313–344.

[31] A. Matreka and M. Strzelecki, *Texture Analysis Methods - A Review*, no. European Cooperation in Science and Technology, COST B11, 1998, pp. 90–924.

[32] N. G. Nguyen, R. S. Poulsen, and C. Louis, *Some New Color Features and Their Application to Cervical Cell Classifacation"*, Pattern Recognition **16** (1983), no. 4, 401–411.

[33] M. S. Nixon and A. S. Aguado, *Feature Extraction and Image Processing*, Newnes, 2002.

[34] P. P. Ohanian and R. C. Dubes, *Performance Evaluation for four Class of Texture Features*, Pattern Recognition **25** (1992), no. 8, 819–833.

[35] T. Ojala and M. Pietikaine, *Texture classification*, Tech. report, University of Oulu.

[36] M. Ozden and E. Polat, *Image Segmentation Using Color and Texture Features*, European Signal Processing Conference (2005).

[37] M. A. Roula, J. Diamond, A. Bouridane, P. Miller, and A. Amira, *A Multispectral Computer Vision System for Automatic Grading of Prostatic Neoplasia*, IEEE Int. Symp. on Biomedical Imaging (2002), 193–196.

[38] F. Samadzadegan, M. J. Valadan Zoj, and M. Kiavarz Moghaddam, *Fusion of GIS Data and High-resolution Satellite Imagery for Post-Earthquake Building Damage Assessment*, Geographic Information Systems.

[39] M. Schroder, M. Schrder, and A. Dimai, *Texture Information in Remote Sensing Images: A Case Study*, Workshop on Texture Analysis, 1998.

[40] L. Semler and L. Dettori, *A comparison Wavelet-Based and Ridgelet-based Texture Classification of Tissues in Computed Tomography*, Advanced in Computer Graphics and Computer Vision **4** (2007), 240–250.

[41] A. Shahbahrami, J. Y. Hur, B. H. H. Juurlink, and S. Wong, *FPGA Implementation of Parallel Histogram Computation*, The 2nd HiPEAC Workshop on Reconfigurable Computing, January 2008, pp. 63–72.

[42] M. A. Tahir, A. Bouridane, and F. Kurugollu, *An FPGA Based Coprocessor for GLCM and Haralick Texture Features and their Application in Prostate Cancer Classification*, Analog Integrated Circuits and Signal Processing **43** (2005), 205–215.

[43] M. A. Tahir, M. A. Roula, A. Bouridane, F. Kurugollu, and A. Amira, *An FPGA Based co-processor for GLCM Texture Features Measurement*, The 10th IEEE Int. Conf. on Electronics, Circuits and Systems **3** (2003), 1006–1009.

[44] M. Tuceryan and A.K. Jain, *Texture analysis*, The Handbook of Pattern and Computer Vision (2nd Edition), 1998, pp. 207–248.

[45] S. E. Umbaugh, *Computer Imaging: Digital Image Analysis and Processing*, Taylor and Francic Group, 2005.

[46] K. Valkealahti and E. Oja, *Reduced Multidimensional Co-Occurrence Histograms in Texture Classification*, IEEE Tran on Patts.ern Analysis and Machine Intelligence **PAMI** (1980), no. 12, 5–45.

[47] J.S. Weszka, C.R. Dyer, and A. Rosenfeld, *A Comparative study of Texture Measures for Terrain Classification*, IEEE Trans. on Systems, Man and Cybernetics **6** (1976), 269–285.

[48] J. Zhang and T. Tan, *Brief review of Invariant Texture Analysis Methods*, Pattern Recognition **35** (2002), 735–747.