Accelerating Texture Features Extraction Algorithms using FPGA Architecture

Ali Reza Akoushideh
Electronic Group
Shahid Chamran Institute of Technology
Rasht, Iran
akushide@afr.ac.ir

Asadollah Shahbahrami
Department of Computer Engineering
University of Guilan
Rasht, Iran
shahbahrami@guilan.ac.ir

Abstract—**Statistical texture features extraction algorithms can be classified into first, second, and higher order. The difference between these classes is that the first-order statistics estimate properties of individual image pixel values, while in the second and higher order statistics estimate properties of two or more image pixel values occurring at specific locations relative to each other. The most popular second order statistical texture features are derived from the co-occurrence matrix, which has been proposed by Haralick. However, the computation of both matrix and extracting texture features are very time consuming. In this paper we improve the performance of those algorithms using FPGA implementations. Our experimental results show that the computational time of co-occurrence matrix is 6.74 times faster than the computational time of extracting thirteen texture features for an image size 128×128 and 8bit gray level. Furthermore whole execution of both algorithms is almost 214x faster than the software implementations.**

Keywords-Haralick texture feature; Image processing; FPGA

I. Introduction

Statistical texture features extraction algorithms represent textures based on the distributions and relationships between image pixels. These algorithms can be classified into first- second, and higher-order. 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 [1]. This is because this approach was recently shown to outperform wavelet packets (a transform based technique) when applied to texture classification [2].

However, the computation of this matrix is very time consuming. Haralick's texture feature calculation can be divided into two parts. First is the calculation of co-occurrence matrices and second is the calculation of texture features using the calculated co-occurrence matrices. These steps are very time consuming. For example, for an image of size 5000×5000, time required is approximately 350 seconds using Pentium 4 machine. The calculation of the co-occurrence matrices and Haralick texture features take 75% and 19% of the total time, respectively [3].

The objective of this work is to improve the performance of the Haralick's texture features using FPGA implementation. FPGAs are reconfigurable device and have ability to execute many complex computations in parallel. These abilities enable a hardware system dedicated to performing fast co-occurrence matrix computations and computing thirteen texture features in parallel.

This paper is organized as follows. We discuss the explanation of the Haralick's texture features extraction algorithms in Section II followed by some important related work in Section III. Our proposed hardware architecture and modified texture feature formula are described in section IV. Details of speedup technique are described in Section V. FPGA selection roles such as used area, and speed have been explained in Section VI. In Section VII, we compare our proposed architecture with microprocessor. Finally, the conclusions are drawn in Section VIII.

II. Haralick Texture Features

Haralick's texture features extraction algorithms can be divided into two parts [8]. First is the calculation of the co-occurrence matrices and second is the calculation of texture features using the calculated co-occurrence matrices. These parts are discussed in the following subsections.

A. *Gray Level Co-occurrence Matrix (GLCM)*

In 1979, Haralick [8], defined the co-occurrence matrix as a second order histogram statistics and it is one of the best known texture analysis algorithms. This algorithm is known as Gray Level Co-occurrence Matrix (GLCM). 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^o, 45^o, 90^o, 135^o) as shown in Fig 1.

Second, image pixels are separated by distance d in eight directions (0^o, 45^o, 90^o, 135^o, 180^o, 225^o, 270^o, 315^o) [9], [1]. The distances can be 1 and 2 pixels based on angles (θ) is chosen between 0^o to 360^o. For classification the fine textures small values of d is required, whereas coarse textures require large values of *d*. In other words, the GLCM is a square matrix with N_g dimension, where N_g is the number of gray levels in image. Each element of the matrix is the number of occurrence of the pair of pixels with value *i* and value *j* which are at distance *d* [10], [11], [12].

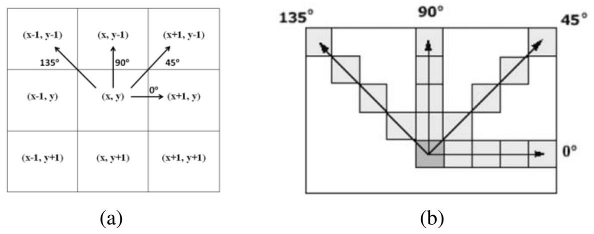

Figure 1. (a)Nearest neighborhood of the point (x, y) and directions of co-occurrence calculation of GLCMs, (b)co-occurrence can be considered also for a greater distance between pairs of points.

Fig. 2 depicts the C implementation of the calculation of co-occurrence matrix with consideration of one direction within a distance *d*.

```
void CoOccurrence_Matrix_Calculation_4() {
for (i=1; i<=N; i++)
  for (j=1; j<=M; j++)
  {
    cooccurance[img[i][j]][img[i][j+d]]++;
    cooccurance[img[i][j]][img[i-d][j]]++;
    cooccurance[img[i][j]][img[i-d][j+d]]++;
    cooccurance[img[i][j]][img[i-d][j-d]]++;
  }}
```

Figure 2. The computation of co-occurrence matrix in four directions of adjacency with distance *d*.

B. Texture Features

The following statistical properties are calculated from the co-occurrence matrix.

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

$$F1 = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} np(i,j)^2 \quad (1)$$

Where $np(i,j)$ *is* the (i,j)th element of the normalized GLCM.

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.

$$F2 = \sum_{N=1}^{Ng-1} (N^2 \times \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} np(i,j)_{|i-j|=N}) \quad (2)$$

3) Correlation (COR): Correlation calculates the linear dependency of the gray level values in the co-occurrence matrix [13]. It shows how the reference pixel is related to its neighbor.

$$Corr = F3 = \frac{\sigma_{xy}}{\sigma_x \times \sigma_y} \quad (3)$$

Where:

$$\mu_x = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} (i \times np(i,j)) \quad (4)$$

$$\mu_y = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} (j \times np(i,j)) \quad (5)$$

$$\sigma_y^2 = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} ((j - \mu_y)^2 \times np(i,j)) \quad (6)$$

$$\sigma_x^2 = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} ((i - \mu_x)^2 \times np(i,j)) \quad (7)$$

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

$$F4^2 = Var^2 = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} ((i - \mu)^2 \times np(i,j)) \quad (8)$$

Where:

$$\mu = \frac{\sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} np(i,j)}{Ng \times Ng} \quad (9)$$

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

$$F5 = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} \frac{np(i,j)}{1-|i-j|^2} \quad (10)$$

6) Sum Average (mean)

$$F6 = \sum_{k=0}^{2Ng-2} (k \times P_{x+y}(k)) \quad (11)$$

Where:

$$P_{x+y}(k) = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} np(i,j)_{|i+j|=k} \quad (12)$$

$$\text{that } k = 0..(2 \times Ng - 2)$$

7) Sum Variance

$$F7 = \sum_{k=0}^{2Ng-2} ((k - F8)^2 \times P_{x+y}(k)) \quad (13)$$

8) Sum Entropy

$$F8 = -\sum_{k=0}^{2Ng-2} (P_{x+y}(k) \times \log (P_{x+y}(k))) \quad (14)$$

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

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

$$F9 = -\sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} (np(i,j) \times \log(np(i,j))) \quad (15)$$

The high entropy image has a great contrast from one pixel to its neighbor and can not be compressed as a low entropy image which has a low contrast [10].

10) Difference Variance

$$F10^2 = \sigma_{P_{x-y}}{}^2 = \sum_{i=0}^{Ng-1} \left(P_{x-y}(k) - \mu_{P_{x-y}}\right)^2 \quad (16)$$

Where:

$$P_{x-y}(k) = \sum_{i=0}^{Ng-1} \sum_{j=0}^{Ng-1} np(i,j)_{|i-j|=k} \quad (17)$$

$$\text{that } k = 0..Ng - 1$$

$$\mu_{P_{x-y}} = \frac{\sum_{k=0}^{Ng-1} P_{x-y}(k)}{Ng} \quad (18)$$

11) Difference Entropy

$$F11 = -\sum_{k=0}^{Ng-1}(P_{x-y}(k) \times \log(P_{x-y}(k))) \quad (19)$$

12) Information Measures of Correlation 1

$$F12 = \frac{HXY - HXY1}{\max\{HX, HY\}} \quad (20)$$

Where HXY is Entropy and:

$$HXY1 = -\sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}(np(i,j) \times \log\left(p_x(i) \times p_y(j)\right)) \quad (21)$$

$$p_y(k) = \sum_{i=0}^{Ng-1}(i \times np(i,k)) \quad (22)$$

$$p_x(k) = \sum_{j=0}^{Ng-1}(j \times np(k,j)) \quad (23)$$

$$HXY2 = -\sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}(p_x(i) \times p_y(j) \times \log\left(p_x(i) \times p_y(j)\right)) \quad (24)$$

$$HX = -\sum_{i=0}^{Ng-1}(p_x(i) \times \log(p_x(i))) \quad (25)$$

$$HY = -\sum_{i=0}^{Ng-1}(p_y(i) \times \log\left(p_y(i)\right)) \quad (26)$$

13) Information Measures of Correlation

$$F13^2 = 1 - e^{-2(HXY2 - HXY)} \quad (27)$$

III. Realated Work

Bariamis et al. [4] presented a hardware implementation to calculate 16 co-occurrence matrices and 4 feature vectors using a single core. The implemented hardware exploited both the symmetry and the sparseness of the matrix. They chose Ng=32, the number of gray level for different image sizes from 512×512 to 2048×2048. In order to use floating point operations to calculate texture features, they used integer arithmetic. Their architecture has 16 co-occurrence matrix computation units. Each input image is divided into different block sizes from 8×8 to 256×256 and is loaded into the corresponding RAM bank. Each pixel is represented using 25 bits, which includes five 5-bit of neighboring pixels at 0^o, 45^o, 90^o, and 135^o directions.

Iakovidis et al. [5] presented an FPGA architecture for parallel computation of 16 co-occurrence matrices (d = 1, 2, 3, 4) and ($\theta = 0^o, 45^o, 90^o, 135^o$) that exploits both their symmetry and sparsenedd. They used 5-bit gray-level representation the same as [4]. Tahir et al. [3] presented a FPGA architecture that computes the co-occurrence matrices of multispectral images. The computation of the co-occurrence matrices is performed by one FPGA core, while the computation of the texture features is performed by a second core that is subsequently programmed onto the FPGA. However, the use of this second core results in a time overhead for reprogramming the FPGA, affecting the overall feature extraction performance.

Bariamis et al. [6] calculated both the co-occurrence matrices and features in hardware, while a significant part of the computations relies on software. Their design in [4] was capable to compute co-occurrence matrices features in hardware, but employed data redundancy in order to achieve high processing throughput. However, the redundancy led to high memory capacity requirements and redundant transfers of data over the PCI bus. They have proposed an FPGA implementation for real-time extraction of co-occurrence matrices texture features from video frames in [7].

In addition, some researchers have used the GPUs in order to improve the performance of the texture features extraction algorithms [14]. They have implemented four texture features, while in medical image classifications usually more texture features are used [15].

IV. Hardware *Implementation*

A. Modification of Features for implementation in FPGA

Processing of floating point data in FPGA takes long time and requires high area in devices. So we modified Haralick's formula to integer format. Consequently we can extract features with FPGA. When we need normalized features, we convert those results to normalized format. Modifications of Haralick's texture features for implementing in FPGA and normalized coefficients, which convert them to floating point, are depicted in Table I.

TABLE I. Modified Haralick Features Formula and Normal Coefficients

Modified Haralick formula	**Normalized coefficient**
$\widehat{F1} = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}(p(i,j))^2$	$\frac{1}{R^2}$
$\widehat{F2} = \sum_{N=1}^{Ng-1}(N^2 \times \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1} p(i,j)_{\lvert i-j \rvert = N}$	$\frac{1}{R}$
$\widehat{F3}^2 = \frac{\widehat{\sigma_{xy}}^2}{\widehat{\sigma_x}^2 \times \widehat{\sigma_y}^2}$	R^3
$\widehat{F4}^2 = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}(Ng^2 \times i - R)^2 \times p(i,j)$	$\frac{1}{R \times Ng^4}$
$\widehat{F5} = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}\frac{k \times p(i,j)}{1 - \lvert i-j \rvert^2}$	$\frac{1}{k \times R}$
$\widehat{F6} = \sum_{k=0}^{2Ng-2} k \times \widehat{P_{x+y}(k)}$	$\frac{1}{R}$
$\widehat{F8} = \sum_{k=0}^{2Ng-2}\widehat{P_{x+y}(k)} \times \left(\log(R) - \log\left(\widehat{P_{x+y}(k)}\right)\right)$	$\frac{1}{R}$
$\widehat{F7} = \sum_{k=0}^{2Ng-2}\left(R \times k - \widehat{F8}\right)^2 \times \widehat{P_{x+y}(k)}$	$\frac{1}{R^3}$
$\widehat{F9} = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1} p(i,j) \times (\log(R) - \log(p(i,j)))$	$\frac{1}{R}$
$\widehat{F11} = \sum_{k=0}^{Ng-1}\widehat{P_{x-y}(k)} \times \left(\log(R) - \log\left(\widehat{P_{x-y}(k)}\right)\right)$	$\frac{1}{R}$
$\widehat{F10}^2 = \sum_{i=0}^{Ng-1}(Ng \times \widehat{P_{x-y}}(k) - \widehat{\mu_{P_{x-y}}})^2$	$\frac{1}{R^2 \times Ng^2}$
$\widehat{F12} = \frac{\widehat{HXY} - \widehat{HXY}1}{\max\{\widehat{HX}, \widehat{HY}\}}$	1
$\widehat{F13^2} = \widehat{HXY}2 - R \times \widehat{HXY}$	$1 - \exp\left(-2 \times \frac{\widehat{F13}}{R^2}\right)$

We used some formulas which are used in the modified Haralick's texture feature. Some of them are followed as:

$$SUM = R = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1} p(i,j) \quad (28)$$

$$np(i,j) = \frac{p(i,j)}{R} \quad (29)$$

$$\widehat{\mu_x} = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1} i \times p(i,j) \quad (30)$$

$$\widehat{\sigma_x}^2 = \sum_{i=0}^{\mathrm{Ng}-1}\sum_{j=0}^{\mathrm{Ng}-1}(R\times i - \widehat{\mu_x})^2 \times p(i,j) \quad (31)$$

$$\widehat{P_{x+y}(k)} = \sum_{i=0}^{\mathrm{Ng}-1}\sum_{j=0}^{\mathrm{Ng}-1} p(i,j)_{|i+j|=k} \quad (32)$$

$$\widehat{p_x(k)} = \sum_{j=0}^{Ng-1} j\times p(k,j) \quad (33)$$

$$\widehat{HX} = \sum_{i=0}^{\mathrm{Ng}-1}\widehat{p_x(\iota)}\times(\log(R)-\log(\widehat{p_x(\iota)})) \quad (34)$$

B. *Architecture*

In order to implement GLCM and 13 features on Virtex series of Xilinx FPGA, we propose a new architecture which is depicted in Fig. 3. The FPGA architecture consists of GLCM core that calculate and sends GLCM in parallel to internal RAM modules via bus interface unit. After this step, GLCM core sends the ready signal to all feature extraction modules immediately. Feature extraction modules start their process in parallel.

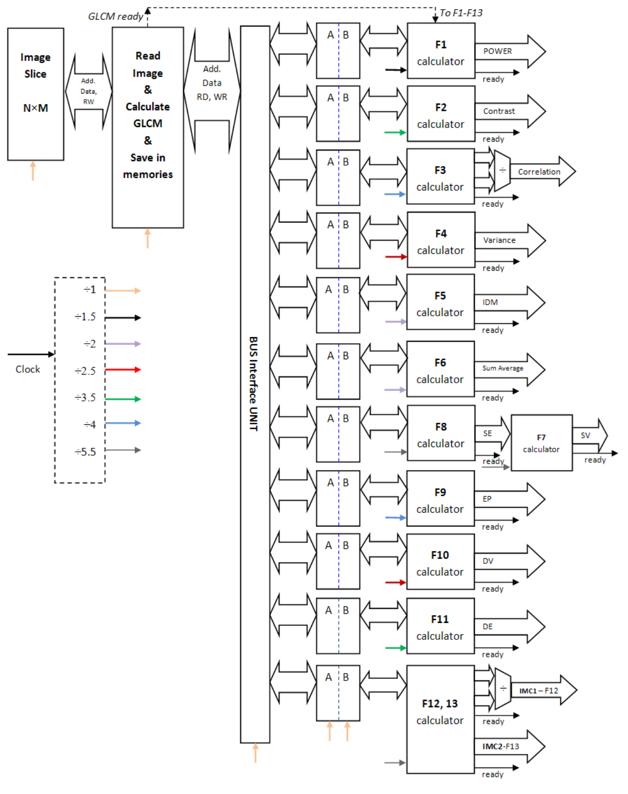

Figure 3. The proposed architecture of Haralick's texture feature extraction on FPGA.

In the proposed architecture we used 16-bit calculation tools and the RAMs are dual ports which have 14-bit address bus and 16-bit data bus. Their code was written with VHDL and synthesized by ISE tools. In order to explain the proposed architecture, we show the block diagrams of GLCM calculator and some feature calculators in fig. 4 and fig. 5, respectively.

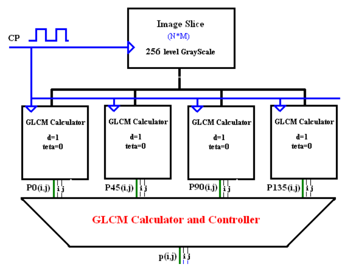

Figure 4. Block diagram of GLCM calculator.

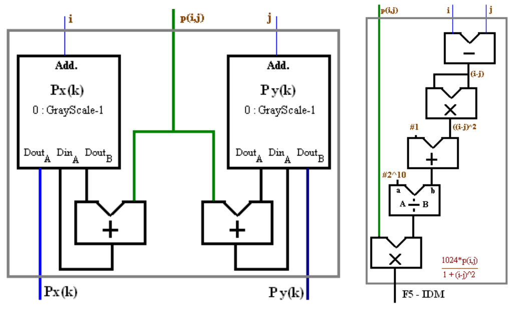

Figure 5. Architecture of $\widehat{p_x(k)}$, $\widehat{p_y(k)}$ and $\widehat{F5}$.

Table II shows number of clock cycle that modules need for calculation of GLCM and features. If we suppose image size is N×M (=128×128), d=1, θ=0° and Ng=256 then we calculate GLCM in 1+3×(N×(M-d)) clock cycles. The features of F3, F12 take more time than other features. In addition, GLCM takes less time than other modules in this configuration. Fig. 6 shows this comparison.

TABLE II. COMPARISON OF GLCM AND FEATURES IN CLOCK CYCLE

step	Feature Modules		Number of Clock cycle	
1	***GLCM***	***Co-Oc. Matrix***	1+3×(N×(M-d))	48769
2	***F1***	***power***	1+Ng×Ng	65537
3	***F2***	***Contrast***	1+ Ng×Ng	65537
4	***F3***	***Corr^2***	1+ 2×Ng×Ng+n	131105
5	***F4***	***Variance***	1+ 2×Ng×Ng	131073
6	***F5***	***IDM***	1+ Ng×Ng	65537
7	***F6***	***SA***	1+ Ng×Ng+2×Ng-1	66048
8	***F8***	***SV***	1+ Ng×Ng+2×Ng-1	66048
9	***F7***	***SE***	1+ Ng×Ng+4×Ng-2	66559
10	***F9***	***EP***	1+ Ng×Ng+2×Ng-1	66048
11	***F10***	***DV^2***	1+ Ng×Ng+2×Ng	66049
12	***F11***	***DE***	1+ Ng×Ng+Ng	65793
13	***F12***	***IMC1***	1+ 2×Ng×Ng+n	131105
14	***F13***	***IMC2***	1+ 2×Ng×Ng	131073

V. SPEEDUP FEATURE EXTRACTION MODULES

Because of different internal components in modules, they have different speeds. If we use these modules all together, the slowest module will affect on the speed of the other modules. In order to speedup feature extraction modules clock dividing technique was proposed.

A. *Single Clock Algorithm*

We select a FPGA from Xilinx Virtex-5 Series and after synthesis by ISE synthesis software version 11.1 we attained speed of each module individually. After synthesis of all modules, the speed fall in the least speed (T=22ns).

This amount of speed can be calculated using equations of (35) and (36). For calculation of total time we have to add GLCM time with maximum total time of modules. We multiply number of modules clock cycle to the least period. After that we add maximum time to GLCM's processing time.

$$T_{total} = t_{GLCM} + \max\{t_{F1} + \cdots + t_{F13}) \quad (35)$$

$$t_{modules} = max.\ period \times modules\ Clock\ Cycle \quad (36)$$

B. *Multiple Clock Dividing Algorithm*

In order to optimize the performance of calculations clock dividing technique is used. Number of clock divider outputs is related to speed of each module. At first we calculate speed of each module individually, then we calculate relation of it with the fastest modules and attain best dividing coefficient.

Table III shows rounded ratio of modules period related to the fastest of them. If we use clock divider with output dividing as 1.5, 2, …, 5.5, and applying of these outputs as clock cycle of modules, we have maximum speed in processing time after synthesis. Consequently, we can speedup final calculation time.

TABLE III. ORIGINAL PERIOD AND MODIFIED PERIOD OF MODULES

Features	Period(ns)	Rounded Ratio	New period (ns)	Number of CP
GLCM	3.998	1	3.998	48769
F1	4.669	1.5	5.997	65537
F2	13.346	3.5	13.994	65537
F3	16.109	4	15.993	131105
F4	8.931	2.5	9.996	131073
F5	7.850	2	7.996	65537
F6	7.896	2	7.996	66048
F7	20.630	5.5	21.990	66048
F8	20.630	5.5	21.990	66559
F9	15.154	4	15.993	66048
F10	8.496	2.5	9.996	66049
F11	13.663	3.5	13.994	65793
F12	21.977	5.5	21.990	131105
F13	21.977	5.5	21.990	131073

The results are attained for an Image with size 128×128 and 8bit gray level.

Equation (37) depicts how to calculate the total processing time using new technique.

$$t_{modules} = New\ period \times Number\ of\ module's\ \mathrm{CP} \quad (37)$$

We compared both techniques for Xilinx FPGA, Virtex 5, FX130T -3 with image properties as N=128, M=128, θ=0°, d=1, Ng=256. In single clock, total processing time was 3,955.46μs and in multiple clocks this time was reduced to 3078.00μs.

VI. FPGA SELECTION

In order to select the optimum FPGA devices, we must calculate sum of required capacity of GLCM and features modules in memory, LUT, IO block and etc. Table IV depicts size of every module in Slice register, Slice LUT, IO pin and DSPs.

TABLE IV. REQUIRED CAPACITY OF EACH MODULE

Modules	Slice Reg.	Slice LUT	IOB	DSP
GLCM	102	163	6	0
F1	123	252	21	1
F2	147	334	33	5
F3	439	902	33	34
F4	134	238	17	2
F5	115	260	17	1
F6	8851	6670	17	1
F7-8	16619	12888	17	10
F9	16563	12671	17	2
F10	4519	3560	17	1
F11	8387	7176	16	1
F12-13	16676	14056	49	14
SUM	**72675**	**59170**	**260**	**72**

After summation, Slice LUT, IO pin we see that we have to select only Virtex series of Xilinx FPGA, because they have internal memory Block, and we approximately need 72675 memory-Slice, 59170 LUT slice and 260 IO pin. We could candidate only XC4VL200 from Virtex-4 series, but we have many options in Virtex-5 as XC5VLX330T, XC5VFX130T and any FPGA from Virtex-6.

Our synthesis did not support virtex-6 series, so we focused on Virtex-4 and Virtex-5 series and we compared them based on Frequency, Slice register and LUT, IOB and DSP. Table V shows final results of these type comparison.

TABLE V. COMPARISON OF XILINX FPGA DEVICES IN CAPACITY, FREQUENCY AND SPEED

Xilinx FPGA Device	Frequency (MHz)	Slice Reg.	Slice LUT	IOB	DSP
Virtex 4 vlx200 -11	168.481	75620	91341	260	72
		42.44%	51.26%	27.08%	75.00%
Virtex 5 fx130T -3	250.112	72675	59170	260	72
		35.05%	28.53%	30.95%	31.30%
Virtex 5 330XT -1	201.039	73343	59494	260	72
		35.37%	28.69%	27.08%	37.50%
Virtex 5 330XT -2	236.102	73531	61192	260	72
		35.46%	29.51%	27.08%	37.50%

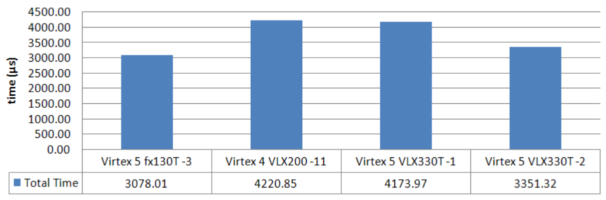

Figure 6. Diagram of total processing time for different FPGA

VII. COMPARISON WITH MICROPROCESSOR

In order to provide a rough comparison with software implementation, we implemented the mentioned algorithms on a Celeron 2.66GHz CPU and FPGA-based implementations for an image size 128×128 and 8-bit gray level and we saw the highlight difference of two methods. The clock speed of the FPGA architecture was 250MHz and the occupied area (% of slices) was about 30%, while the clock speed for Intel Celeron was 2660MHz. Table VI shows this comparison on taking time of each module individually.

TABLE VI. COMPARISON OF MODULES SPEED AND PC INDIVIDUALLY

Modules	Celeron 2.66GHz.	Virtex 5 fx130T -3	ratio
GLCM	005.00 ms	0.19 ms	25.64
F1	012.70 ms	0.39 ms	32.31
F2	240.60 ms	0.92 ms	262.35
F3	012.00 ms	2.10 ms	5.72
F4	195.70 ms	1.31 ms	149.37
F5	009.50 ms	0.52 ms	18.13
F6	466.60 ms	0.53 ms	883.47
F7-8	653.56 ms	1.45 ms	449.98
F9	653.50 ms	1.46 ms	446.49
F10	15.60 ms	1.06 ms	14.77
F11	245.90 ms	0.66 ms	372.47
F12-13	432.30 ms	0.92 ms	469.54

Image size: 128×128; Gray Level: 8 bit

Also Table VII depicts over all comparison of the execution time between a microprocessor based system and FPGA-based. As this table shows the FPGA implementation is almost 214x faster than software implementation on a microprocessor based system even though the PC platform has a clock speed which is about 10 times faster than FPGA device. This performance improvement is mainly due to parallel implementations of feature extraction algorithms.

TABLE VII. COMPARISON OF PROPOSED ARCHITECTURE TO PC

	Celeron 2.66GHz	Virtex 5 fx130T -3
Execution Time(ms)	658.556	3.078
Speed up Factor	1×	214×

Image size: 128×128; Gray Level: 8 bit

VIII. CONCLUSIONS

Haralick's texture feature extraction algorithm can be divided into two parts. First is the calculation of the co-occurrence matrices and second is the calculation of texture features using the calculated co-occurrence matrices. Both of these algorithms are computationally intensive. In this paper a parallel architecture was proposed to calculate co-occurrence matrix and thirteen texture features in parallel with reconfigurable devices which they are useful in many applications intensive problems such as image processing. We targeted the Virtex-5 XC5FX130-3 device. All hardware modules have been synthesized, placed, and routed using Xilinx ISE tool. Experimental results show that the computational time of extracting texture features is almost 6.74x more than the computational time of the co-occurrence matrix. In addition, the total execution time of Haralick's texture features is almost 214x faster than the software implementation on a microprocessor based system.

REFERENCES

[1] A. Matreka and M. Strzelecki, "Texture Analysis Methods - A Review," European Cooperation in Science and Technology, COST B11, 1998, pp. 90–924.

[2] K. Valkealahti and E. Oja, "Reduced Multidimensional Co-Occurrence Histograms in Texture Classification," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, pp. 5–45, 1980.

[3] 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, vol. 43, pp. 205–215, 2005.

[4] D. Bariamis, D. K. Iakovidis, and D. E. Maroulis, "Dedicated Hardware for Real-Time Computation of Second-Order Statistical Features for High Resolution Images," Lecture Notes in Computer Science, vol. 4179, pp. 67–77, November 2006.

[5] D. K. Iakovidis, D. E. Maroulis, and D. G. Bariamisa, "FPGA Architecture for Fast Parallel Computation of Co-occurrence Matrices," Microprocessors and Microsystems, vol. 31, no. 2, pp. 160–165, March 2007.

[6] D. G. Bariamis, D. K. Iakovidis, D. E. Maroulis, and S. A. Karkanis, "An FPGA-based Architecture for Real Time Image Feature Extraction," Proc. 17th Int. Conf. on Pattern Recognition, 2004.

[7] D. Maroulis, D. K. Iakovidis, and D. Bariamis, "FPGA-based System for Real-Time Video Texture Analysis", Journal of Signal Processing Systems, vol. 53, no. 3, pp. 419–433, December 2008.

[8] R. M. Haralick, K. Shanmugam, and I. Denstien, "Textural Features for Image Classification"," IEEE Trans. on Systems, Man and Cybernetics, vol. SMC-3, no. 6, pp. 610–621, November 1973.

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

[10] M. V. Boland, "Quantitative Description and Automated Classification of Cellular Protein Localization Patterns in Fluorescence Microscope Images of Mammalian Cells," Ph.D. dissertation, University of Pittsburgh, 1999.

[11] 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, vol. 31, March 2006.

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

[13] A. B. M. A. Tahir and F. Kurugollu, "Accelerating the Computation of GLCM and Haralick Texture Features on Reconfigurable Hardware," Int. Conf. on Image Processing, vol. 5, pp. 2857–2860, 2004.

[14] M. Gipp, G. Marcus, N. Harder, A. Suratanee, K. Rohr, R. König, R. Männer, "Accelerating the Computation of Haralick's Texture Features using Graphics Processing Units (GPUs)", Proc. of the World Congress on Engineering 2008.

[15] M. Gipp, G. Marcus, N. Harder, A. Suratanee, K. Rohr, R. König, R. Männer, "Haralick's Texture Features Computed by GPUs for Biological Applications", IAENG International Journal of Computer Science, 36:1, IJCS_36_1_09.