An FPGA-based Architecture for Real Time Image Feature Extraction

D.G. Bariamis, D.K. Iakovidis, D.E. Maroulis
University of Athens
Dept. of Informatics and Telecommunications
Illisia, 15784 Athens, Greece
rtsimage@di.uoa.gr

S. A. Karkanis
Technological Educational Institute of Lamia
Dept. of Informatics and Computer Technology
3rd klm Old Nat. Road, 35100 Lamia, Greece
sk@teilam.gr

Abstract

We propose a novel FPGA-based architecture for the extraction of four texture features using Gray Level Cooccurrence Matrix (GLCM) analysis. These features are angular second moment, correlation, inverse difference moment, and entropy. The proposed architecture consists of a hardware and a software module. The hardware module is implemented on Xilinx Virtex-E V2000 FPGA using VHDL. It calculates many GLCMs and GLCM integer features in parallel. The software retrieves the feature vectors calculated in hardware and performs complementary computations. The architecture was evaluated using standard grayscale images and video clips. The results show that it can be efficiently used in realtime pattern recognition applications.

1 Introduction

Realtime image pattern recognition is a challenging task which involves image processing, feature extraction and pattern classification. It applies to a wide range of applications including multimedia, military and medical ones. Its high computational requirements force systems to use very expensive clusters, custom VLSI designs or even both. These approaches suffer from various disadvantages, such as high cost and long development times. Recent advances in fabrication technology allow the manufacturing of high density and high performance Field Programmable Gate Arrays (FPGAs) capable of performing many complex computations in parallel while hosted by conventional computer hardware.

A variety of architecture designs capable of supporting realtime pattern recognition have been proposed in the recent literature, such as implementations of algorithms for image and video processing [4, 5], classification [5, 6] and image feature extraction algorithms [7, 8]. Although texture plays a significant role in image analysis and pattern recognition only a few architectures implement on-board textural feature extraction. Most prominent approaches include the extraction of Gabor wavelet features for face/object recognition [7] and the computation of mean and contrast Gray Level Cooccurrence Matrix (GLCM) features [8]. In the second case the two features are approximated without computing GLCMs.

In this paper we propose a novel FPGA-based architecture for realtime GLCM texture analysis. The proposed architecture combines both software and hardware to raster scan input images with sliding windows and produce 16-dimensional feature vectors consisting of four GLCM features calculated for four directions.

2 Methods

The image feature extraction process involves raster scanning the image with windows (subimages) of a given dimension and a given scanning step. This step corresponds to the offset between two consecutive subimages.

Cooccurrence matrices encode the gray level spatial dependence based on the calculation of the 2^{nd} order joint conditional probability density function $f(i, j, d, \theta)$, which is computed by counting all pairs of pixels at distance d having gray levels i and j at a given direction θ. We have considered four directions, namely $0°$, $45°$, $90°$ and $135°$, as well as a predefined distance of one pixel in the formation of the matrices. Among the 14 statistical measures originally proposed [1], we have considered four, namely angular second moment (f_1), correlation (f_2), inverse difference moment (f_3) and entropy (f_4). These four measures provide high discrimination accuracy [3], which can be only marginally increased by adding more features in the feature vector.

$$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_{ij}^2 \quad (1)$$

Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04)
1051-4651/04 $ 20.00 IEEE

$$f_2 = (\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} i \cdot j \cdot p_{ij} - \mu_x\mu_y)/(\sigma_x\sigma_y) \quad (2)$$

$$f_3 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} \frac{1}{1+(i-j)^2} p_{ij} \quad (3)$$

$$f_4 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_{ij} \cdot log p_{ij} \quad (4)$$

where p_{ij} is the ijth element of the normalized cooccurrence matrix, N_g is the number of gray levels of the image, μ_x, μ_y, σ_x and σ_y are the means and standard deviations of the marginal probabilities $P_x(i)$ and $P_y(i)$, obtained by summing the rows or columns of the p_{ij} matrix.

In order to simplify the hardware implementation and increase software performance, the floating point operations (Eq. 1-4) were replaced by integer operations (Eq. 5-9),

$$f_1 = (\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij}^2)/r^2 \quad (5)$$

$$f_2 = \left(r \cdot N_g^2 \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} i \cdot j \cdot c_{ij} - r^2 \right) \cdot \frac{N_g - 1}{S} \quad (6)$$

$$S = \sum_{k=1}^{N_g} \left(r - C_x(k) \right)^2 \quad (7)$$

$$f_3 = 2^{-30} \cdot \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij} \cdot \mathrm{IDMLUT}[i-j] \quad (8)$$

$$f_4 = 2^{-26} \cdot \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} c_{ij} \cdot (\mathrm{LOGLUT}[c_{ij}] - 2^{26} \cdot \log r) \quad (9)$$

where c_{ij} is the ijth 16-bit integer element of the unnormalized cooccurrence matrix and r is a 16-bit integer normalizing constant. $C_x(k)$ is an array of 16-bit integers that represent the sum of each column of the GLCM. The IDMLUT[k]=$\lfloor 2^{30}/(1+k^2) \rfloor$ and LOGLUT[k]=$\lfloor 2^{26} \cdot \log k \rfloor$ arrays correspond to 32-bit look up tables used for the approximation of $1/(1+k^2)$ and $\log k$ respectively. N_g, i and j are represented using 8-bit integers.

3 Architecture Description

The proposed architecture consists of two stages, a preprocessing stage and the feature extraction block (Fig. 1). The first prepares input data to be processed by the feature extraction block while the second combines both software and hardware to calculate GLCM features. Most of the GLCM feature vectors are calculated in hardware. Software supports hardware by performing supplementary computations.

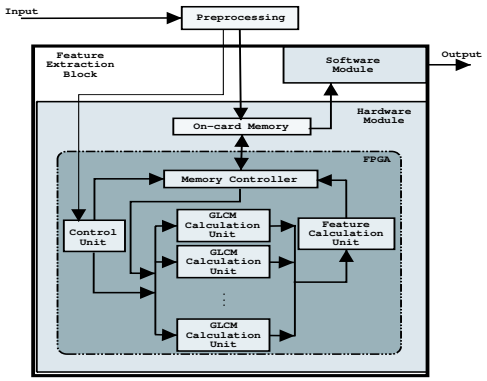

Figure 1. Overview of the architecture

3.1 Preprocessing

The preprocessing handles the conversion of the image into an array A suitable for processing by the feature extraction block. Each element $\overline{a} = [a_0 a_1 a_2 a_3 a_4]$ of A corresponds to each pixel. It is formed by five integers. The first (a_0) is the gray level of the corresponding pixel, while the others (a_1, a_2, a_3, a_4) are the gray levels of its first neighbors for the four directions considered. Following a linear quantization of the image intensities to 64 gray levels, they can be adequately represented by 6-bit integers. This results in a 30-bit representation of each element that can be read by the hardware in one cycle, thereby enabling the simultaneous computation of the GLCM intensity transitions for the four angles.

3.2 Feature Extraction Block

The feature extraction block consists of a hardware and a software module. The hardware module is implemented on a Xilinx Virtex-E V2000 FPGA using VHDL [9]. The FPGA is hosted by the Celoxica RC1000 card that also includes four 2MB static RAM banks. The FPGA block RAM and the distributed RAM implemented on chip sum up to 1Mbit. The host computer preprocesses the image, and loads the resulting array to one of the four memory banks on the card. The FPGA calculates the feature vectors and stores them in another bank, from where they are retrieved by the host. The FPGA architecture consists of:

- A control unit that coordinates the functionality of the FPGA, by generating the signals that synchronize the other units
- A memory controller that handles transactions from and to the on-card memory
- A parallel array of 72 GLCM calculation units, organized in 18 GLCM calculation unit quadruplets
- A feature calculation unit capable of reading GLCMs,

calcualting the feature vectors and storing them into the on-card memory.

3.2.1 GLCM Calculation Units

The GLCM calculation unit consists of a 16-way set associative array [2] with a capacity of 512 cells and the circuitry needed for GLCM calculation.

GLCM calculation units receive pairs of gray level values as input. These pairs are obtained by decoding the element $\overline{a}$ of array A. The GLCM calculation units of any GLCM quadruplet are fed by a_0a_1, a_0a_2, a_0a_3 and a_0a_4 pairs. When a gray level pair a_0a_i, $i \in 1, 2, 3, 4$ is read by the GLCM calculation unit, the respective cell of the set associative array is incremented by one.

3.2.2 Feature Calculation Unit

The feature calculation unit receives as input a GLCM generated by each GLCM calculation unit and outputs a vector $\overline{V} = [V_1, V_2, V_3, V_4, V_S]$ to the on-card memory.
$V_1 = \sum c_{ij}^2$
$V_2 = \sum i \cdot j \cdot c_{ij}$
$V_3 = \sum c_{ij} \cdot \mathrm{IDMLUT}[i-j]$
$V_4 = \sum c_{ij} \cdot (\log c_{ij} - \log r)$
$V_S = \sum \left(r - C_x(k)\right)^2$
where $\log c_{ij}$ and $\log r$ are represented as fixed point values. $\log r$ is precalculated in software and stored in a register before use, while IDMLUT[k] (8) is stored in a 64×32-bit ROM.

Several methods have been proposed in the literature for logarithm calculation, including Newton-Raphson and power series. These are iterative methods which require either a long pipeline and many computational units or few computational units and many cycles to calculate a single result. We implemented an efficient approach for the calculation of $\log c_{ij}$, which has limited requirements in hardware resources and results in a low approximation error. It consists of 3 steps:
1. The integer part of $\log c_{ij}$, $l_i = \lfloor \log c_{ij} \rfloor$ is calculated by means of a priority encoder [9]. Its value equals to the index of the most significant non-zero bit of c_{ij}.
2. The fractional part of $\log c_{ij}$, $l_f = \log c_{ij} - \lfloor \log c_{ij} \rfloor$ is calculated by a single shifter. The shifter displaces c_{ij} by $(16 - \lfloor \log c_{ij} \rfloor)$ bits to the left.
3. The fractional part l_f is transformed as follows:

$$l'_f = \begin{cases} l_f + 1/4 \cdot l_f & \text{if} \quad l_f \leq 1/2 \\ l_f - 1/4 \cdot l_f + 1/4 & \text{if} \quad l_f > 1/2 \end{cases} \qquad (10)$$

The first two steps result in a linear approximation of the logarithm between values corresponding to $c_{ij} = 2^n, n \in$N, while the third contributes to an increase of the approximation accuracy.

3.2.3 Software Module

The software module retrieves the vectors V from the on-card memory. The integer components of each vector are converted into 32-bit floating point values. Substituting these values in Eq. 5-9 the corresponding GLCM features are calculated. Moreover the software module is capable of supporting the calculation of vectors V that were not computed in hardware. The software implementation used is highly optimized, based on an algorithm that takes advantage of the sparseness and symmetry of the cooccurrence matrix.

4 Results

The performance of the proposed architecture was evaluated using standard grayscale still images and videos. The images used were *DocCageCity.0002* (I_1) and *GroundWaterCity.0000* (I_2) from the Vistex database and *Lenna* (I_3), having 256×256 dimensions. The video clip used was the *Silent* (I_4), with a frame dimension of 128×128 pixels and a total of 300 frames. In Fig. 2 the three still images and one frame of the video are shown, as well as the distribution of non-zero elements in the cooccurrence matrices produced for each image.

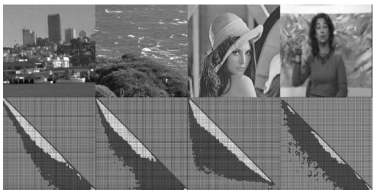

Figure 2. Test images and cooccurrence matrices' distribution

The hardware implementation calculates all results using integer-only arithmetic. For f_1, f_2 and f_3 this results in no error cumulation and output identical to the optimized software implementation. For f_4, the logarithm is calculated through approximation. This method (Sec. 3.2.2) produces results that have a relative error of less than 0.5% for every 16-bit input, except some distinct cases. Using a small 16×32-bit ROM, we can eliminate these cases and claim a maximum relative error of 0.5%.

The host processor used was an Athlon 1GHz and the FPGA functions at 25MHz. The feature vectors are calculated rather fast, but the factor that determines total performance is the density of the cooccurrence matrices produced by the image. If the 512 non-zero GLCM element limit is exceeded, the corresponding feature needs to be calculated in software, resulting in an overhead. For the images and

Table 1. Experimental results

	W	S	t_1	r_s (%)	t_2	t_H	t_S
I_1	8	4	30.8	100.0	0.0	30.8	189.4
	8	8	9.89	100.0	0.0	9.89	51.6
	16	4	46.2	100.0	0.0	46.2	286.7
	16	8	13.9	100.0	0.0	13.9	65.6
	32	4	70.8	98.65	10.5	81.3	301.4
	32	8	20.3	98.65	2.7	23.0	85.6
I_2	8	4	30.8	100.0	0.0	30.8	196.8
	8	8	9.89	100.0	0.0	9.89	54.4
	16	4	46.2	100.0	0.0	46.2	303.2
	16	8	13.9	100.0	0.0	13.9	69.7
	32	4	70.8	99.84	1.2	72.0	297.4
	32	8	20.3	99.84	0.3	20.6	85.5
I_3	8	4	30.8	100.0	0.0	30.8	186.8
	8	8	9.89	100.0	0.0	9.89	51.2
	16	4	46.2	100.0	0.0	46.2	297.4
	16	8	13.9	100.0	0.0	13.9	61.5
	32	4	70.8	97.48	19.6	90.4	342.9
	32	8	20.3	97.48	5.1	25.4	95.9
I_4	8	4	5.7	100.0	0.0	5.7	50.5
	8	8	1.9	100.0	0.0	1.9	14.2
	16	4	6.6	100.0	0.0	6.6	65.4
	16	8	2.3	100.0	0.0	2.3	20.5
	32	4	8.9	89.05	15.5	24.4	97.6
	32	8	2.7	89.05	4.4	7.08	27.9

the video tested, the computation times are shown in Table 1, where W and S are the window and step parameters in pixels, t_1 and t_2 are the time in ms needed for the calculation in the hardware and software module respectively, and r_s column is the ratio of vectors successfully computed in hardware versus those computed in software. t_H is the total time in ms needed for the extraction of the feature vectors for each image. The last column of the above tables (t_S) presents the corresponding execution times of the software module without any hardware acceleration. The results shown for the video clip (I_4) are the average times for all of its frames.

5 Conclusions

We proposed and implemented a novel FPGA-based architecture for realtime extraction of four GLCM features. It combines both hardware and software to achieve accurate feature calculation as well as a considerably high performance, which can be accounted to the efficient parallel implementation. Most of the calculations are performed in hardware. The results show that the proposed architecture could be efficiently used for real time pattern recognition applications. Realtime operation depends on the image size (D), the window size (W) and the scanning step (S) and can be guaranteed for particular combinations of these parameters (e.g. $D \leq 256$, $W \leq 16$ and $S \geq 4$).

6 Acknowledgments

This work was realized under the framework of the Operational Program for Education and Vocational Training Project Pythagoras cofunded by European Union and the Ministry of National Education of Greece.

References

[1] R.E. Haralick, K. Shanmugam, I. Dinstein, *Textural Features for Image Classification*, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-3, No. 6, Nov 1973

[2] J.L. Hennesy & D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann, May 2002

[3] D.E. Maroulis, D.K. Iakovidis, S.A. Karkanis, D.A. Karras, *CoLD: a versatile detection system for colorectal lesions in endoscopy video frames*, Computer Methods and Programs in Biomedicine, vol 70, no. 2, pp. 99-186, Elsevier Science, 2003

[4] W. Luk, P. Andreou, A. Derbyshire, F. Dupont-De-Dinechin, J. Rice, N. Shirazi and D. Siganos, *Field-Programmable Logic: From FPGAs to Computing Paradigm*, Springer-Verlag, Berlin, 1998

[5] M. Nibouche, A. Bouridane, D. Crookes, O. Nibouche, *An FPGA-based wavelet transforms coprocessor*, in Proc. IEEE Int. Conf. Image Processing, pp. 194-197, vol.3, 2003.

[6] H. Hikawa, *Implementation of Simplified Multilayer Neural Network with On-chip Learning*, Proc. of the IEEE International Conference on Neural Networks (Part 4), Vol. 4, 1999, pp 1633-1637.

[7] T. Nakano, T. Morie, and A. Iwata, *A Face/Object Recognition System Using FPGA Implementation of Coarse Region Segmentation*, SICE Annual Conference 2003, pp. 1418-1423, Fukui, Aug. 4-6, 2003.

[8] K. Heikkinen and P. Vuorimaa, *Computation of Two Texture Features in Hardware*, Proceedings of the 10th International Conference on Image Analysis and Processing, Venice, Italy, pages 125-129, September 27-29, 1999.

[9] K.C. Chang, *Digital Design and Modeling with VHDL and Synthesis*, IEEE Computer Society Press - Wiley, 1997