# An efficient implementation of GLCM algorithm in FPGA

M Atitallah, Rostom Kachouri, H. Mnif, M Kammoun

# An efficient implementation of GLCM algorithm in FPGA

M.A. Ben Atitallah
LETI (E.N.I.S.), University of Sfax, TUNISIA
National Engineering School of Gabes (ENIG)
mohamed.amine@esiee.fr

R. Kachouri
Gaspard Monge Computer Science Laboratory
ESIEE-Paris
University Paris-Est Marne-la-Vallée, FRANCE
rostom.kachouri@esiee.fr

M. Kammoun
LETI (E.N.I.S.), University of Sfax, TUNISIA
National Engineering School of Sfax (ENIS)
manelkammounenis@gmail.*com*

H.Mnif
LETI (E.N.I.S.), University of Sfax, TUNISIA
ENET'com Sfax, TNISIA
hassene.mnif@enetcom.usf.tn

***Abstract*—This paper presents hardware (HW) architecture for fast parallel computation of Gray Level Co-occurrence Matrix (GLCM) in high throughput image analysis applications. GLCM has proven to be a powerful basis for use in texture classification. Various textural parameters calculated from the GLCM help understand the details about the overall image content. However, the calculation of GLCM is very computationally intensive. In this paper, 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. This architecture was implemented on a Xilinx Zedboard and Virtex 5 FPGAs using Vivado HLS. The performance is then compared against other implementations. The validation results show an optimization on the order of 33% in latency number by contribution to the literature implementation.**

***Keywords—Image analysis applications; Symmetric Co-occurrence matrix (GLCM); Parallel computation; Haralick's texture feature; Hardware Implementation; FPGA; Vivado HLS; Optimization; Latency number***

## I. INTRODUCTION

The methods of statistical analysis of the texture [1] are divided into two groups: a first-order analysis such as mean, variance, etc ..., and a second-order analysis as the co-occurrence matrix. The purpose of texture analysis is to extract the characteristic properties of the object in a complex image and to express them in a feature vector form. The representation obtained will serve as a basis for the subsequent steps. The second-order analysis is the most popular method to detect the texture [2] because the results obtained by those of the first order are not significant. In fact, the second-order statistics take into account the spatial distribution of two pixels thus describing the relation on the neighborhood of the pixels.

Texture analysis through GLCM matrices is an essential step for image analysis applications such as medical imaging [3], recognition of character for complex image [4] and agronomic or industrial applications. However, the computation of GLCM matrices are very time consuming. Therefore, methods to accelerate their computations are highly desired. In order to improve the performance of co-occurrence matrices algorithm, we propose an architecture on FPGA platform. In the proposed architecture, the co-occurrence matrix is computed in parallel. We have implemented the proposed architecture on Zedboard Zynq-7000 FPGA device. The HW/SW implementation on FPGA shows an optimization on the order of 33% in latency number by contribution to the literature implementation.

This paper is organized as follows. We discuss the explanation of Gray level co-occurrence matrix algorithm (GLCM) in Sect 2 followed by some important related work in Sect 3. Our proposed hardware architecture is described in Sect 4. Experimental results are discussed in Sect 5. Finally, the conclusions are drawn in Sect 6.

## II. GRAY LEVEL CO-OCCURRENCE MATRIX (GLCM)

Haralick's texture features extraction algorithms [5] 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. For this research project we are just interested in the GLCM matrix calculation.

The matrices of co-occurrence [6] have become the most known and the most used to extract the texture feature. They estimate properties of images relating to second-order statistics. A co-occurrence matrix measures the probability of appearance of pairs of pixel values located at a distance in the image. This algorithm is known as 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 $\theta$ as an orientation angular.

The angular directions θ conventionally used are 0, 45, 90 and 135 degrees. The neighborhood relations between pixels, necessary for calculating the matrices, are illustrated in Figure 1. The matrices obtained according to the four directions are calculated as indicated in (1), (2), (3) and (4).

$$P(i,j,\mathrm{d},0) = \left|\left\{\left((k,l),(m,n)\right) \in (N*M)^2 \; while \; \left(k-m=0, |l-n|=\mathrm{d}, \mathrm{I}_{\mathrm{k,l}}=i, \mathrm{l}_{\mathrm{m,n}}=j\right)\right\}\right| \quad (1)$$

$$P(i,j,\mathrm{d},45) = \left|\left\{\begin{matrix}\left((k,l),(m,n)\right) \in (N*M)^2 while \\ (k-m=\mathrm{d}, \mathrm{l}-\mathrm{n}=-\mathrm{d}) \vee (k-m=-\mathrm{d}, \mathrm{l}-\mathrm{n}=\mathrm{d}), \mathrm{I}_{\mathrm{k,l}}=i, \mathrm{l}_{\mathrm{m,n}}=j\end{matrix}\right\}\right| \quad (2)$$

$$P(i,j,\mathrm{d},90) = \left|\left\{\left((k,l),(m,n)\right) \in (N*M)^2 \; while \left(|k-m|=\mathrm{d}, l-n=0, \mathrm{I}_{\mathrm{k,l}}=i, \mathrm{l}_{\mathrm{m,n}}=j\right)\right\}\right| \quad (3)$$

$$P(i,j,\mathrm{d},135) = \left|\left\{\begin{matrix}\left((k,l),(m,n)\right) \in (N*M)^2 \; while \\ (k-m=\mathrm{d}, \mathrm{l}-\mathrm{n}=\mathrm{d}) \vee (k-m=-\mathrm{d}, \mathrm{l}-\mathrm{n}=-\mathrm{d}), \mathrm{I}_{\mathrm{k,l}}=i, \mathrm{l}_{\mathrm{m,n}}=j\end{matrix}\right\}\right| \quad (4)$$

With :

• (k, l) are the coordinates of a graylevel pixel i ∈ [0, nmax-1].

• (m, n) those of the graylevel pixel j ∈ [0, nmax-1].

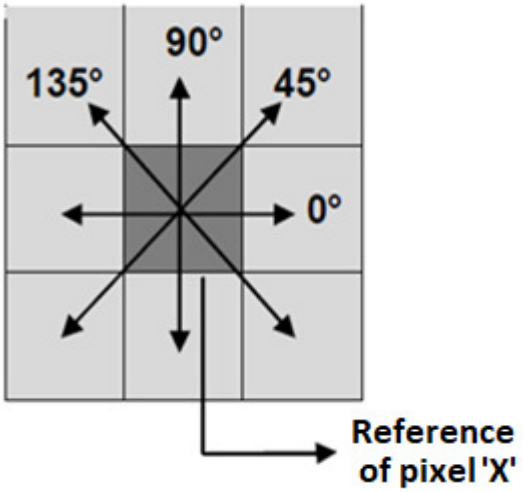


Fig. 1. Closest neighbors of pixel 'x' in four directions

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 0 | 1 | 3 | 2 |
| 0 | 2 | 3 | 2 |
| 1 | 2 | 3 | 0 |

P(i, j,1, 0°)= (3 on top, 2 on left)

$$\begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 0 & 2 & 1 \\ 1 & 2 & 0 & \textcircled{4} \\ 1 & 1 & 4 & 0 \end{pmatrix}$$

P(i ,j,1, 45°)=

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 5 & 0 \end{pmatrix}$$

P(i, j,1, 90°)=

$$\begin{pmatrix} 4 & 2 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 1 & 6 & 0 \\ 1 & 1 & 0 & 4 \end{pmatrix}$$

P(i ,j,1,135°)=

$$\begin{pmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$

Fig. 2. Example of co-occurrence matrices built from a 4 × 4 image composed of 4 gray levels

The Figure 2 shows an example of calculating P (i, j) from a small image 4 × 4 composed with four gray levels (0, 1, 2, 3). This example is limited to the case d = 1 and θ= 0. The element (2, 3) of the matrix P (1, 0) is equal to 4, this means that there are four configurations in the image where a pixel of gray level 2 is separated horizontally from another pixel of gray level 3 by a distance 1. These configurations are shown in gray lines in the image (Figure 2).

## III. RELATED WORKS

Many researchers have been working on accelerating the process of computation the GLCMs algorithms on FPGAs platforms.

Girisha and al [7] proposed a HW implementation to compute the GLCM matrix for (θ=0°) and (d=1). The proposed architecture is implemented on a Xilinx XC2VP30 platform programmed in VHDL. It is characterized by 13696 Slices and 136 KB RAM block. The input image of this architecture is 8x8 pixels, each pixel is represented on 4 bits (Ng=8).

The input image is recorded in two different memory blocks to determine the pixel and the neighboring pixel. In this architecture Girisha et al used a comparator to detect the end of line. This architecture consumes a lot of memory space because the input image is stored in two memory blocks. In this case the large images can't be implemented. Girisha has just implemented a GLCM matrix for a single angle.

To solve the problems of Girisha, Ali Reza and al [8] proposed an architecture making it possible to calculate the four matrices GLCM of the four angles θ (0 °, 45 °, 90 °, 135 °) in parallel for images of size 128X128 pixels, the computation of GLCM is done for pixels with 8 bits in which (Ng=256) and (d=1). So the size of the GLCM is 256x256 pixels. Each coefficient in this matrix is on 16 bits. The proposed architecture is implemented on a Xilinx Virtex 5 using VHDL language. This architecture makes it possible to calculate and send the GLCMs in parallel to the internal RAM modules via a bus interface unit. After this step, the GLCM core immediately sends the ready signal to all texture feature modules. The texture feature modules start their process in parallel. The calculations are done on the basis of an integer format of 16 bits. The RAMs used in this architecture is dual port to have the ability to read and write data at the same time.

This architecture has as disadvantage that the calculating texture is in an integer format, which generates

inaccurate results. The size of the image is fixed and the distance of the matrix GLCM is also fixed.

The objective of our research project is to propose and implement a hardware architecture more efficient than the architecture proposed by Ali Reza.

## IV. HARDWARE ARCHITECTURE OF GLCM

FPGA circuits have emerged as a privileged target platforms to implement intensive signal processing applications [9]. For this reason several academic and industrial efforts have been devoted in order to increase the productivity of FPGA-based designs by means of using High Level Synthesis (HLS) tools. HLS approach in Electronic Design Automation (EDA) is a step in the design flow aiming at moving the design effort to higher abstraction levels [10]. This evolution towards HLS-based methodologies can be easily traced along the history of hardware system design [11]. Although the first generations of HLS tools failed to produce efficient hardware designs, different reasons have motivated researchers to continue improving these tools.

In this context Xilinx propose a new HLS tool allows to automatically translate a code written in high level language like c / c ++ or system c into an Register Transfer Language (RTL) code like the Figure 3 shows. Once the IP has been designed, it can be integrated into the IPs library of Vivado IDE and thus used for the construction of the hardware design.

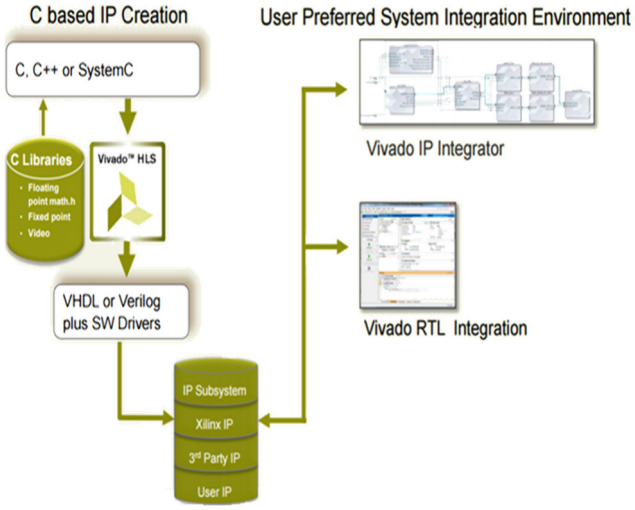


Fig. 3. Vivado HLS conception flow

### A. Architectural design for GLCM

The GLCM hardware architecture is illustrated in Figure 4. Our architecture was developed by a c code compatible with the HLS tool. The proposed GLCM architecture is implemented for a 128x128 image. Each pixel of the image is represented by (n=8) bits, so (Ng=256) because $Ng = 2^n$. Our IP reads each pixel, calculates the GLCM for the four directions (θ=0°, 45°, 90° and 135°) and for distance (d=1), and sends them to the software (SW) part. Our IP has two outputs, one presents the GLCM results and one to detect the end of sending data.

In fact, our architecture makes it possible to calculate the GLCM of the four directions in parallel in order to accelerate the treatment. The size of GLCM is NgxNg. The GLCM coefficients are presented on (m=16) bits.

The SW part is processed by the ARM cortex A9 processor at 667 MHz. It is used to send and receive data through the DMA "Direct Access Memory". In addition, in our HW design we use a DMA in order to speed the Data transfer.

### B. Optimizations of GLCM architecture

In this section, we are going to explore the possible optimization steps that could be done in order to achieve an efficient hardware implementation. The C code was written in HLS-friendly syntax with neither file read/write, nor dynamic memory allocation nor system calls. The optimization steps are incrementally applied to the design as listed in Table 1. From our point of view, a fair comparison between designs is valid only for adjacent rows in order to observe the impact of adding this optimization to the overall design performance.

We have developed different optimizations for the implementation of the GLCM matrices for the four angles in order to select the optimal optimization. First, in the first optimization "Optimization 1", we ran our code without adding any options just with default HLS optimization. In the second optimization "Optimization 2", we added in addition to the "Optimization 1" the UNROLL directive at the level of the loops of the GLCM matrices. Finally in the third optimization "Optimization 3", we added to the "Optimization 1" the PEPELINE directive at the level of the loops of the GLCM matrices. Table 1 shows the implement results of all optimizations in terms of area and latency number using Zynq-7000 FPGA Device.

TABLE I. EXPERIMENT RESULTS USING DIFFERENT OPTIMIZATIONS

| Architecture | FPGA | DSP % | FF % | LUT % | BRAM % | Latency number |
|---|---|---|---|---|---|---|
| #Optimization 1 | Zynq-7000 | 0 | 0 | ~ 0 | 97 | 82180 |
| #Optimization 2 | Zynq-7000 | 0 | 11 | 40 | 97 | 33156 |
| #Optimization 3 | Zynq-7000 | 0 | 14 | 41 | 97 | **32774** |

In fact, the synthesis of the three optimizations which were developed with the Vivado_HLS tool shows that the "Optimization 3" presents the best performances. Indeed, thanks to the use of pipeline in the "Optimization 3", we managed to reduce the latency number up to 76% compared to "Optimization 1". We admit the "Optimization 3" since it ensures a compromise of latency number but with increase in the area.

In the next section we present our experiments, discuss the obtained results and prove the performance of our proposition.

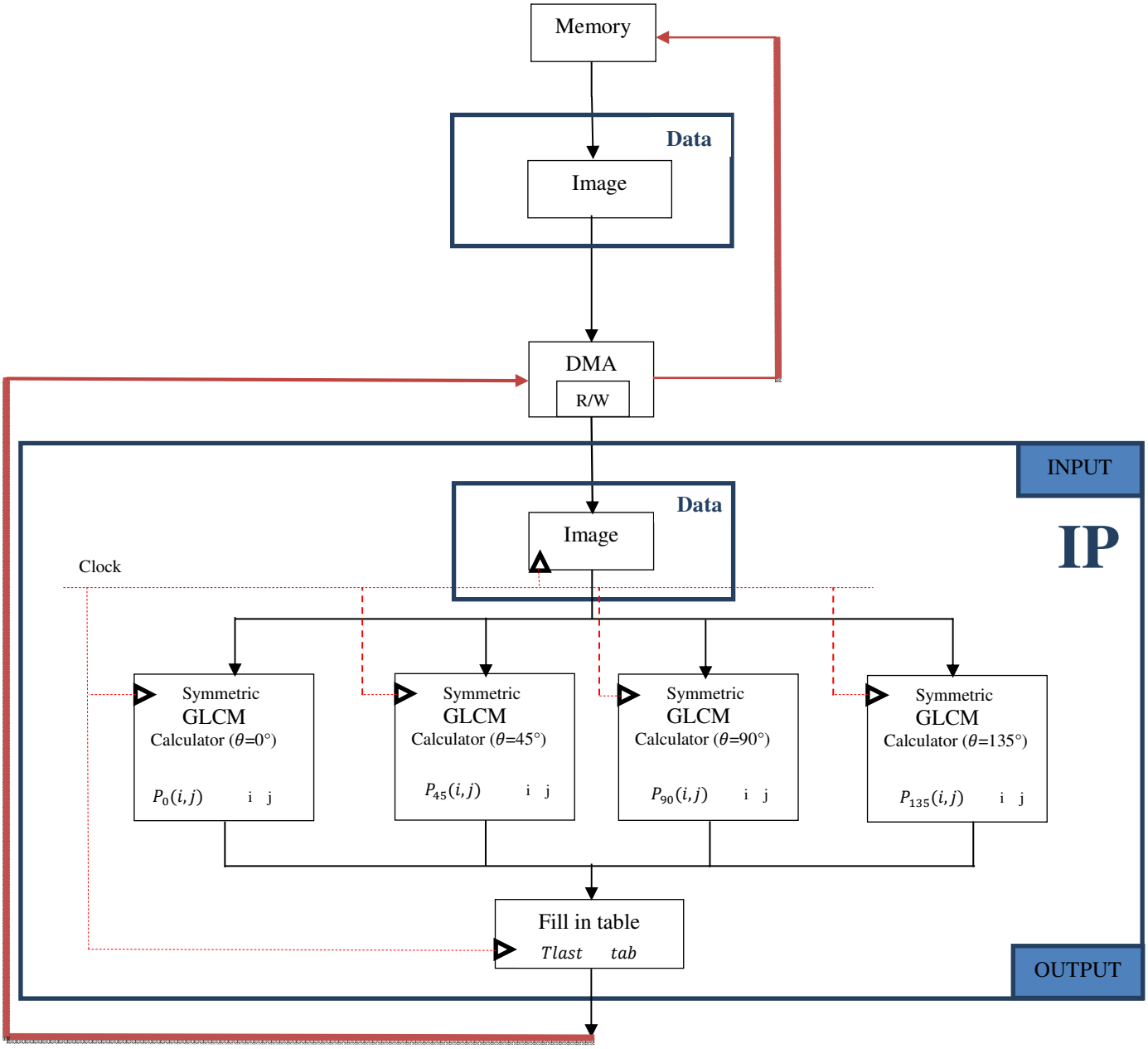


Fig. 4. GLCM HW architecture

## V. EXPERIMENTAL RESULTS

In this part, our goal is to valid the performance of our GLCM architecture in HW/SW context.

The GLCM architecture was implemented using HLS tool (for θ (0 °, 45 °, 90 °, 135 °)). This architecture were then implemented on Zedboard Zynq-7000 FPGA device. The Zedboard card shown in Figure 5 is part of the FPGA family of Series 7 that benefits both performance of the two Artix and Kintex families. It is characterized by 53200 Slices, 560 KB BRAM, with 220 DSP Slices.

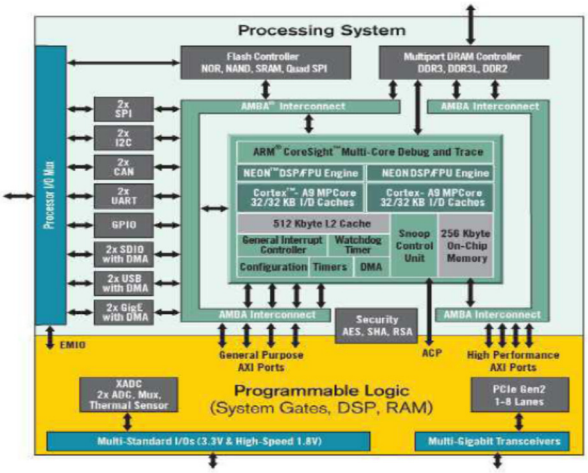


Fig. 5. Architecture of Zedboard FPGA

The Figure 6 shows that the architecture of our system concerns the conception of the hardware design, the software application and the design of the customized IPs. In fact, we start by creating the design from the IPs then validate and synthesize our design, and finally generate the bitstream to configure the hardware part. This is done from the Xilinx Vivado IDE tool. Then we move to the software application designed to program the hardware part of the design. So, in order to accelerate the applications, we will design hardware accelerator to integrate them on the FPGAs part of the circuit via Vivado_HLS[12].

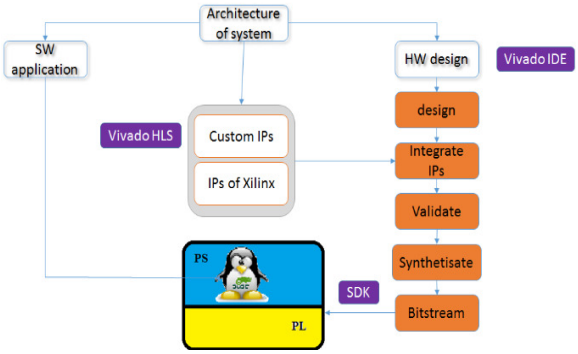


Fig. 6. System design flow

The Xilinx Zedboard-based FPGA platform is an example of such a circuit integrating a dual Cortex A9 processor and a PL of the FPGA Series 7 as shown in Figure 7. It uses a series of protocols to connect the processor with the PL. Inside the Zedboard architecture, the SW is programmed into the processing system (PS) in which we find an ARM cortex-A9 processor operating at a frequency of 667 MHz. In fact, the HLS GLCM architecture has connected to the PS with DMA. The DMA is able to manipulate data for reading or writing via memory.

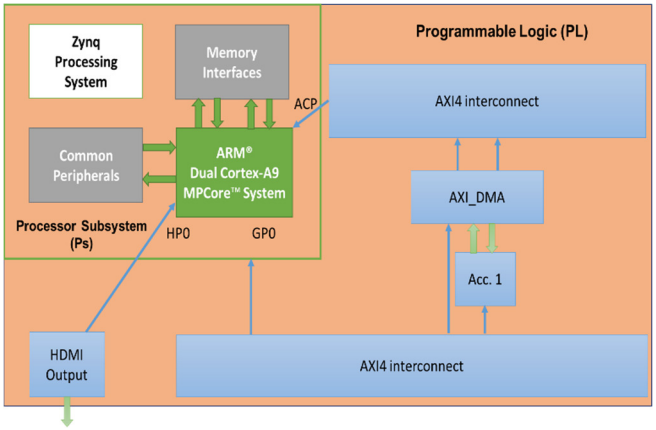


Fig. 7. Heterogeneous SoC system

The Table 2 shows a comparison of the proposed method with the literature.

The Figure 8 shows that our HW / SW architecture has allowed an optimization on the order of 33% in latency number compared to Reza architecture [8]. In fact the treatment is done in 32774 latency number by our method instead of 48769 latency number by Reza method [8].

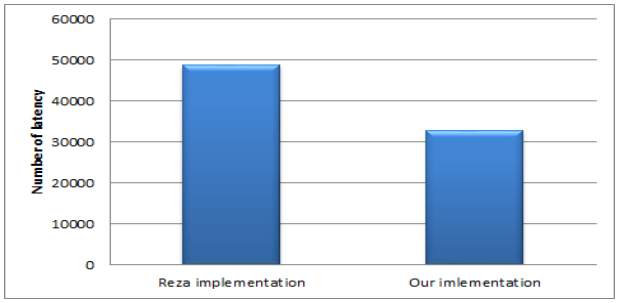


Fig. 8. Comparison of latency number of the two architectures

## VI. CONCLUSION

In conclusion, we proposed a HW / SW implementation of the four GLCM matrices for the four angles θ (0°, 45°, 90°, 135°). This architecture is developed on a Zedboard platform that integrates an ARM Cortex A9 processor. The GLCM design was completely implemented on hardware. The validation results show an optimization on the order of 33% in latency number by contribution to the literature implementation.

Finally as a perspective we are considering to integrate the GLCM architecture in an Optical Character Recognition (OCR).

## ACKNOWLEDGMENT

We thank our colleagues from ESIEE Paris, who provided insight and expertise that greatly assisted the research and improved the manuscript.

TABLE II. PERFORFORMACE OF DIFFERENT PROPOSED ARCHITECTURES

| Proposed architectures | FPGA | Bits number | GLCM size | Image size | HW calculation | Freq (Mhz) | Latency number | Execution time |
|---|---|---|---|---|---|---|---|---|
| Ali Reza [8] | virtex 5 | 8 | 256x256 | 128x128 | GLCM | 250 | 48769 | 0.19 ms |
| Our architecture | virtex 5 | 8 | 256x256 | 128x128 | GLCM | 250 | **32774** | **0.12 ms** |
| Our architecture | Zynq-7000 | 8 | 256x256 | 128x128 | GLCM | 100 | **32774** | **0.39 ms** |

## REFERENCES


[1] G. N. Srinivasan, and Shobha G, "Statistical Texture Analysis," Proceedings of world academy of science, engineering and technology volume 36 december 2008 ISSN 2070-3740.

[2] Shervan Fekri-Ershad, "A Review on Image Texture Analysis Methods," International Online Journal of Image Processing and Pattern Recognition Vol. 1, No.1, pp. 1-63, 2018.

[3] M.A. Tahir, A. Bouridane, and F. Kurugollu, "An FPGA Based Coprocessor for the Classification of Tissue Patterns in Prostatic Cancer," International Conference on Field Programmable Logic and Applications, FPL 2004: Field Programmable Logic and Application pp 771-780 l.

[4] M.A. Ben Atitallah, A. Boudabous, A. Ben Atitallah, R. Kachouri, "Complexity study of the Gamma correction method for text extraction from complex images," 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA, 2015).

[5] A.Dasha, P.Kanungoa, B.P.Mohantyb b, "A Modified Gray level Co-occurrence Matrix based Thresholding for Object Background Classification," Image Analysis & Computer Vision Lab., Procedia Engineering 30 (2012) 85 – 91.

[6] Haralick, Robert M., and Karthikeyan Shanmugam, "Textural features for image classification," IEEE Transactions on systems, man, and cybernetics 6 (1973).

[7] Girisha A B, M C Chandrashekhar, Dr.. M Z kurian, "FPGA implementation of GLCM," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 2, Issue 6, June 2013.

[8] Ali Reza, Asadollah, Babak , "High performance implementation of texture features extraction algorithms using FPGA architecture," J Real-Time Image Proc (2014) 9:141–157.

[9] I. Gonzalez, E. El-Araby, P. Saha, T. El-Ghazawi, H. Simmler, S. G. Merchant, B. M. Holland, C. Reardon, A. D. George, H. Lam, G. Stitt, N. Alam, and M. C. Smith, "Classi_cation of Application Development for FPGA-Based Systems," Aerospace and Electronics Conference, 2008. NAECON 2008.

[10] W. Meeus, K. Van Beeck, T. Goedeme, J. Meel, and D. Stroobandt, "An Overview of Today's High-Level Synthesis Tools," Design Automation for Embedded Systems (2012).

[11] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An Introduction to High-Level Synthesis," IEEE Design Test of Computers (2009).

[12] Declan O'Loughlin, Aedan Coffey, Frank Callaly, Darren Lyons, Fearghal Morgan, "Xilinx Vivado High Level Synthesis," Case studies (2013).