Hardware Accelerator for Feature Extraction based on Circle Views Signature

Huda D. Jomma
Ph.D. researcher at Computer & Systems Eng. Dept., Minia University,
Minia, Egypt.

Aziza I. Hussein
Computer & Systems Eng. Dept., Minia University
Minia, Egypt
Electrical & Computer Eng. Dept, Effat University
Jeddah KSA

Alaa M. Hamdi
Electronics, Communications & Computer Eng. Dept., Helwan university
Cairo, Egypt

ABSTRACT

This paper presents a hardware accelerator for feature extraction based on circle views (CVs) signature suitable for shape recognition. The heavy computational and memory access needs in shape features extraction using general purpose sequential processors enforces the use of a hardware accelerator. This work presents some modifications to the original CVs signature algorithm. These modifications are intended to reduce time and space requirements for the hardware accelerator. A software version for 1NN classifier is implemented based on the modified CVs signature algorithm. This software version records 89.85% recognition rate using MPEG-7 dataset which is too close to that scored by the original CVs signature algorithm. This small reduction in performance can be ignored against the reduction in the hardware requirements and the computation time achieved. A parallel architecture for the hardware accelerator is proposed. Using 16 processing elements, the proposed hardware accelerator achieved 50.34 times speedup compared to the standard software PC implementation. A study is done to measure the effect of changing the number of processing elements on the speedup gained and the hardware components used by the proposed hardware accelerator. The proposed hardware accelerator is implemented in a field programmable gate array (FPGA) by using Verilog hardware description language.

Keywords

Feature extraction, shape recognition, real-time image processing, hardware accelerator, FPGA.

1. INTRODUCTION

Real-time object shape recognition is a difficult task that involves preprocessing the image, extracting shape features and classifying the shape according to stored training set shape features. Object shape recognition has a wide range of applications including multimedia, military and medical science. The recognition process is divided into two parts: creating the features of the training set (the off-line phase) and classifying a query features according to the training set features (the on-line phase). During the on-line phase, real-time feature extraction remains a crucial challenge, since it consumes most of the computation time. So, this paper focuses on the acceleration of features computation. Advances in fabrication technology permits the manufacturing of field programmable gate arrays (FPGAs) having high performance and high density. These FPGAs can be used as powerful computing systems for image processing applications [1, 2].

A variety of architecture designs capable of supporting real-time object shape recognition have been proposed in the literature. Implementations of algorithms for image feature extraction were proposed in [3, 4, and 5]. The work in [3] extracts the Gabor wavelet features for face/object recognition. Heikkinen et. al. [4] presented hardware architectures for computing two texture features: mean and contrast. A novel FPGA-based architecture for the extraction of four texture features using Gray Level Co-occurrence Matrix (GLCM) analysis was proposed in [5]. Hedberg et. al. [6] presented hardware architecture of a labeling algorithm with extraction of simple features: area and center of gravity (COG). The work in [7] presented hardware architecture for computing three basic image component feature descriptors, namely, centre of gravity, area, and bounding box. An FPGA-based accelerator for Fourier descriptors computing for color object recognition was proposed in [8].

Recently, new shape recognition techniques that achieved high recognition rates have been proposed. Some of these methods are 2D histogram representation of shapes called shape contexts (SC) [9], triangle area representation (TAR) [10], a shape tree that is used to represent a hierarchical description [11], inner-distance shape context (IDSC) [12], and the circle views (CVs) signature [13]. Despite the high recognition rates these methods achieved, they need a lot of computations. So, they are in need to hardware accelerators to speed up their computations.

In this paper, a hardware accelerator for computing the circle views (CVs) signature is proposed. The CVs signature aimed to be: very simple, easy to compute, and computationally efficient. In addition, it constitutes a translation, rotation and scale invariant signature [13]. There are many advantages that make the CVs signature algorithm suitable for hardware implementation. The high recognition rate of the CVs signature and the simplicity of the needed calculations are the main reason to choose it. Also, many of these calculations are independent from each other. This will give the chance to design a parallel architecture to compute the CVs signature matrix. Some important modifications are introduced to the original CVs signature algorithm that simplifies the computations needed by the proposed accelerator and thus reduces its FPGA hardware requirements. To test the robustness of the introduced modifications, a software version of the shape classification system is implemented using Matlab that is based on the modified CVs signature algorithm.

The remainder of this paper is organized as follows: Section 2 gives a review on needed algorithms. Section 3 presents the proposed hardware accelerator for computing the CVs signature. Section 4 presents the experimental results for the proposed hardware accelerator. Finally, Section 5 includes the conclusions of the proposed work.

2. REVIEW ON NEEDED ALGORITHMS

In this section, the main algorithms needed to implement the proposed hardware accelerator for computing circle views (CVs) signature will be reviewed. The CVs signature is a shape signature for recognizing 2D object silhouettes. The shape is an important visual feature of a graphical object and also one of the basic features used to describe image content [13]. The CVs signature algorithm extracts the shape features to describe the shape contour. It is obvious that the detection and extraction of the shape contour is a prerequisite for the extraction of shape features. So, contour tracing technique should be executed firstly to detect the external shape contour and produce ordered list of contour pixels. These sequential contour pixels will be used by the CVs signature algorithm to recognize and classify the shape object. In the following subsections, details about these two algorithms will be provided.

2.1 Contour Tracing Algorithm

The goal of the contour tracing algorithm is to extract an external shape contour. Contour tracing finds the start point of an object, traces the shape contour, and extracts the shape contour in the form of an ordered contour pixels list. Many algorithms were proposed for contour tracing. Haig et. al. [14] proposed a simple and efficient contour tracing algorithm. First the algorithm starts to scan the image pixel by pixel in raster scan mode from top to bottom and from left to right until an outer shape contour starting pixel is encountered. Then, the algorithm begins reading the eight adjacent neighbors of the start pixel. The problem now is how effectively to search and find the next contour pixel in the eight adjacent neighbors. The other important point that must be considered is from which neighbor pixel the search among the eight neighbors should start. Each pixel in the eight neighbors is assigned an index as shown in figure 1. Tracing is performed in a clockwise direction. From the contour start pixel P_S, a procedure called tracer is executed to find the next contour pixel starting the search from P_5. The end of contour tracing algorithm will be reached if the tracer identifies P_S as an isolated pixel. If P_S is not an isolated point, the contour pixel following P_S will be output. For any other contour pixel the search in the 8-neighborhood starts from the neighbor d+2, where d is the index of the previous pixel relative to the current pixel. The tracer procedure will be executed repeatedly until a stop condition occurs. The stop condition

P_5	P_6	P_7
P_4	P_C	P_0
P_3	P_2	P_1

Fig 1: The indices of the 8-neighbourhood of the center pixel (P_C)

occurs when the start pixel P_S is visited again and its next pixel is the same as that found in the beginning of the search.

2.2 Circle Views (CVs) Signature Algorithm

The main idea of the CVs signature algorithm is based on creating many views for the shape contour from sampled points on a viewing circle. This circle is centered at the shape centroid. The following algorithm summarizes the steps for constructing the CVs signature matrix [13].

Algorithm

```
Step1:  Input a binary shape image.
Step2:  Extract the shape contour.
Step3:  Sample out $N$ points on the shape contour with $x$ and $y$ coordinates.
        $contour\_Points = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
Step4:  Compute the centroid coordinates $(x_c, y_c)$.
        $x_c = \frac{1}{N}\sum_{u=1}^{N} x(u)$        $y_c = \frac{1}{N}\sum_{u=1}^{N} y(u)$
Step5:  Generate $N$ uniform viewing points on a circular orbit centered at the shape centroid and having a radius $R$.
        $viewing\_Points = \{(x_{v1}, y_{v1}), (x_{v2}, y_{v2}), \ldots, (x_{vN}, y_{vN})\}$
Step6:  Construct an $N$ x $N$ CVs signature matrix as follows:
        for $view$ = 1 to $N$
            get coordinates of viewing point number ($view$)
            $(x_v, y_v) = viewing\_Points(view)$
            $i = view$
            for $k$ = 1 to $N$
                get coordinates of contour point number $i$
                $(x_{cont}, y_{cont}) = contour\_Points(i)$
                  CVs signature ($i$, $view$) =
                $\sqrt{(x_v - x_{cont})^2 + (y_v - y_{cont})^2}$
                $i = (i \% N) + 1$
            end
        end
Step7:  Output the CVs signature matrix.
```

In [13], the features of two viewing circles were merged to capture more information. This obviously enriches the information content of the CVs signature. The Fourier transform was applied to the CVs signature to generate the shape descriptor. The CVs signature achieved 83.71% retrieval rate using bulls-eye test on MPEG-7 database [13].

3. THE PROPOSED HARDWARE ACCELERATOR FOR COMPUTING CVS SIGNATURE

The main purpose of the proposed work is to design a hardware accelerator for computing CVs signature. There is a wide difference between software and hardware implementations. It is a simple task to implement any algorithm in a typical PC environment. The programmer does not take into account the memory requirements, power consumption, and logic usage. These requirements must be considered in advance during hardware system design process. The main bottlenecks regarding the hardware implementation of the CVs signature are the need of tremendous calculations and all these calculations comprise floating point numbers arithmetic's. Since the CVs signature matrix size is 128*128, it needs several multiplications, additions, subtractions, divisions, and square root operations. Thus, some modifications will be proposed for the original CVs signature algorithm that will account for the hardware system design requirements and make it more suitable for hardware implementation. Most of the proposed modifications concentrate on the reduction of the tremendous calculations needed as well as avoiding the usage of floating point numbers arithmetic's. The hardware accelerator for computing CVs signature is implemented on FPGA by using Verilog HDL language. Figure 2 illustrates the main modules of the proposed system. The hardware accelerator comprises four main components, a parallel memory module (storing an input binary image), contour tracer module, contour sampler module, and CVs signature computation module. The detailed description of each module will be presented below.

3.1 Parallel Memory Module

The major bottleneck in the hardware implementation of contour tracing algorithms is the large number of memory accesses needed for tracing the whole shape contour. For each contour pixel, the contour tracer needs to read a 3*3 window that contains the eight adjacent neighbors from the memory that stored the input binary image. Most of the previous work stores the image in a linear RAM array. In this way, nine memory accesses are needed for each window to be read. This will cost the system too much time for only reading sequentially each window of the shape contour pixels. So, achieving real-time performance is difficult due to this memory access demands in contour tracing and hence hardware acceleration is unavoidable.

Ratnayake et. al. in [15] designed a parallel memory architecture by exploiting FPGA memory blocks (BRAMs) that can achieve reading the 3*3 window in one memory access. This obviously will reduce the total memory access time for tracing the whole shape contour by factor of nine. As shown in figure 3, the parallel memory architecture has four hierarchical memory stacks. Each stack consists of four BRAMs constructed as dual port with 1 bit data width and 18K memory location. As shown, the lines of the input image is stored in the parallel memory stacks sequentially, where the first line is stored in stack0, the second line is stored in stack1,the fifth line is stored in stack0 and so on. By the same manner, the pixels of each line is stored sequentially in the block RAMs. This strategy that used for storing the input image facilitates reading any 16 pixels in one memory access [15]. No doubt, incorporating this parallel memory in the proposed hardware implementation will reduce the total execution time needed to compute the CVs signature. So, the parallel memory architecture introduced by Ratnayake to store the input binary image is re-implemented.

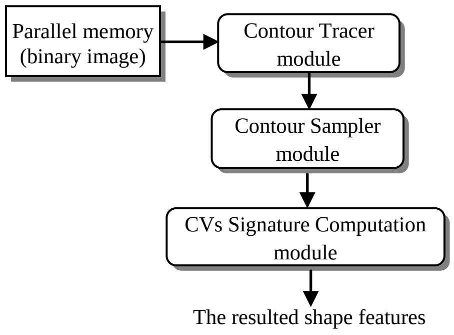

Parallel memory (binary image)
Contour Tracer module
Contour Sampler module
CVs Signature Computation module
The resulted shape features

Fig 2: The main components of the proposed hardware accelerator

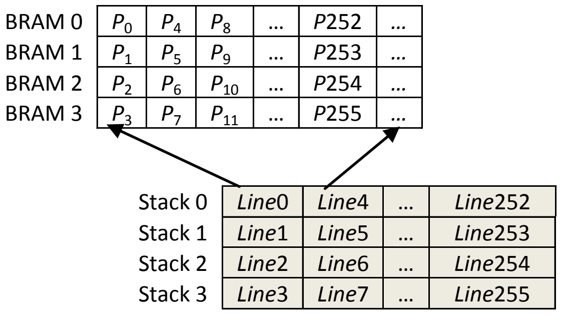

BRAM 0 | P_0 | P_4 | P_8 | … | $P252$ | …
BRAM 1 | P_1 | P_5 | P_9 | … | $P253$ | …
BRAM 2 | P_2 | P_6 | P_{10} | … | $P254$ | …
BRAM 3 | P_3 | P_7 | P_{11} | … | $P255$ | …
Stack 0 | $Line0$ | $Line4$ | … | $Line252$
Stack 1 | $Line1$ | $Line5$ | … | $Line253$
Stack 2 | $Line2$ | $Line6$ | … | $Line254$
Stack 3 | $Line3$ | $Line7$ | … | $Line255$

Fig 3: Hierarchical architecture of the parallel memory (assume image size 256*256)

3.2 Contour Tracer Module

During tracing journey, the contour tracer counts the contour pixels and outputs the number of contour pixels (contour length). Later, the contour sampler module will use the number of contour pixels to compute the sampling step. The coordinates of the shape start pixel are stored in two registers (x_s, y_s). The contour tracer starts tracing the shape contour by the aid of the binary image data stored in the parallel memory module. As shown in figure 4, the contour tracer sends the center pixel coordinates (x_i, y_i) of the window to the parallel memory address generator. The address generator computes the address of each neighboring pixel in the 3*3 window and passes them to the corresponding block RAMs of the parallel memory. The parallel memory returns the neighboring pixels values of the 3*3 window. In the proposed hardware for contour tracer, the tracing algorithm introduced in [14] is implemented to trace the outer contour of the shape. The following steps demonstrate briefly the procedures done by the proposed hardware to implement this algorithm:

1- Initialize the *contour_length* register to 1.

2- Make the current pixel coordinates (x_i, y_i) = (x_s, y_s).

3- Read the 3*3 neighborhood window for the current pixel (x_i, y_i) from the parallel memory in one memory access.

4- Search for the first foreground pixel in the 8-neighborhood (indexed as shown in figure 1) starting from pixel P_5.

5- If no foreground pixel found then

An isolated point is found and contour tracing ends

Else

Store the index of the next pixel in a register named "*next*".

contour_length = *contour_length* + 1

6- Compute the coordinates of the next contour pixel (x_n, y_n) using the lookup tables DXLUT and DYLUT shown in figure 5(a, b). Store the coordinates of this second contour pixel into two registers (x_{s2},y_{s2}).

7- Make current pixel coordinates (x_i, y_i) = (x_n, y_n).

8- Read the 3*3 neighborhood window for the current pixel (x_i, y_i) from the parallel memory in one memory access.

9- Compute d (where d is the index of the previous pixel relative to the current pixel) using the lookup table DLUT shown in figure 5 (c) and the index of the next pixel stored in the register "*next*".

```
10- Search the 3*3 neighborhood window for the first
    foreground pixel starting from the neighbor index $d$+2.
11- Store the index of this foreground pixel into the register
    "next" and compute the coordinates of the next pixel ($x_n$, $y_n$)
    using DXLUT and DYLUT lookup tables.
12- If ($x_i$, $y_i$) = ($x_s$, $y_s$) and ($x_n$, $y_n$) = ($x_{s2}$, $y_{s2}$) then
        Stop contour tracing (stop condition occurred).
    Else
        contour_length = contour_length + 1
        Go to step 7
```

The proposed hardware initializes a 16 bits register named "*contour_length*" to 1. This register will be incremented each time a new contour pixel is found. The two registers facing the parallel memory (x_i, y_i) will be loaded initially by the coordinates of the shape starting pixel (x_s, y_s). The tracer receives the values of 3*3 pixels window, centered at (x_i, y_i), returned from the parallel memory module in one memory access. These values of the eight neighbors will be stored in an 8 bits register named P. So, P_j is the jth neighbor of the center pixel (current pixel). Then, the tracer starts to search for the first foreground contour pixel in the 8-neighborhood pixels. If a foreground pixel is found, the index of this pixel will be stored in a register named "*next*". If the center pixel of the window is the start contour pixel (x_s, y_s), the proposed hardware enforces the search to start from neighbor P_5. For the other contour pixels, the search starts from the neighbor that has the index d+2, where d is the index of the previous pixel relative to the current pixel. A lookup table (DLUT) shown in figure 5(c) is introduced to determine the value d+2 directly. The input for this lookup table is the value stored in the "*next*" register, which is the index of the next pixel in the previous 8-neighborhood. Obviously, this next pixel in the previous 8-neighborhood is the center pixel of the current window. Two lookup tables DXLUT and DYLUT shown in figure 5(a, b) are used to compute the coordinates of this next pixel and then store them in two registers (x_n, y_n). DXLUT and DYLUT lookup tables store the differences in the x and y coordinates of the eight neighboring pixels relative to the window's center pixel. For the next window, the next pixel will be its center pixel, so the coordinates (x_n, y_n) are then transferred to the inputs of the parallel memory module (x_i, y_i) to read the next 3*3 window.

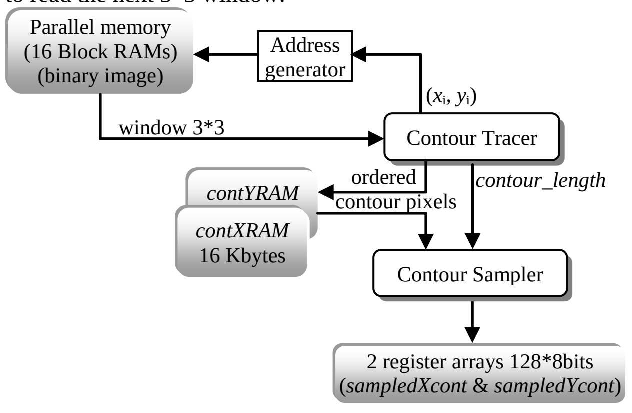
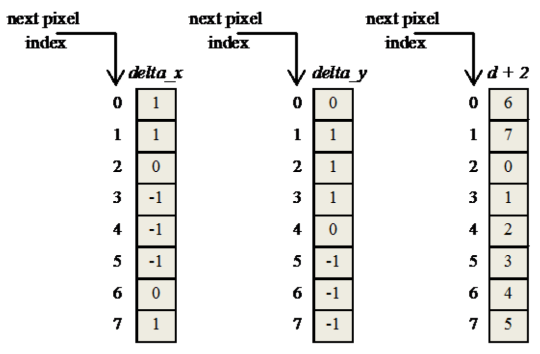


Fig 4: Detailed description for computing the sampled contour points


next pixel index
delta_x
next pixel index
delta_y
next pixel index
d + 2

next pixel index	delta_x	delta_y	d + 2
0	1	0	6
1	1	1	7
2	0	1	0
3	-1	1	1
4	-1	0	2
5	-1	-1	3
6	0	-1	4
7	1	-1	5

(a) DXLUT (b) DYLUT (c) DLUT

Fig 5: Lookup tables contents: (a) DXLUT (b) DYLUT (c) DLUT

The occurrence of the stop condition is monitored by using comparators. These comparators compare the coordinates of the current pixel (x_i, y_i) by the coordinates of the stored start pixel (x_s, y_s) and also compare the coordinates of the next pixel (x_n, y_n) with the coordinates of the stored second contour pixel (x_{s2}, y_{s2}). If these comparators indicate the equivalence of these coordinates, the contour tracer module stops tracing and outputs the number of contour pixels. The contour tracer module stores the ordered shape contour pixel coordinates into two 16 Kbytes block RAMs named *contXRAM* and *contYRAM*.

3.3 Contour Sampler Module

The goal of the contour sampler module is to sample N points out of the whole shape contour pixels. Many systems had chosen the value N = 128 for sampling the contour. This is an essential step before computing the CVs signature. As shown in figure 4, the inputs of the contour sampler are the total number of contour pixels (stored in *contour_length* register) and the coordinates of the ordered contour pixels list stored in the two RAMs (*contXRAM* and *contYRAM*). The output is two register arrays (128*8 bits) named *sampledXcont* and *sampledYcont*. These register arrays will contain the coordinates of the 128 sampled contour points.

To sample the obtained contour, the sample step needs to be computed. The step usually has a fraction. In the proposed system, a method that uses only integer calculations is introduced. This method depends on shifting the contents of the "*contour_length*" register by 7 bits to right. This is equivalent to the division by 128. The 7 bits shifted out are used as the division remainder (R), and the other bits as the integer division result (I). The starting pixel of the contour is considered as the first sample point and its coordinates are stored in the first location of the *sampledXcont* and *sampledYcont* register arrays. During contour sampling, the index of the next sample point "*Next_sample_idx*" will be determined as illustrated below:

```
1- Initialization: $TR$ = 0, Next_sample_idx = 0
2- Next_sample_idx = Next_sample_idx + $I$,
   $TR$ = $TR$ + $R$
3- If ($TR$ >= 128)
       Next_sample_idx = Next_sample_idx + 1
       $TR$ = $TR$ - 128
4- Retrieve a contour pixel coordinates from contXRAM and
   contYRAM stored at Next_sample_idx
5- If the number of sampled contour points <= 128, then
       Go to step 2
```

```
    Else
        Stop
```

Where, *TR* is an accumulator register for the remainder *R*. Each time a new index of the next sample point "*Next_sample_idx*" is updated, the *contXRAM* and *contYRAM* RAMs are accessed at the address "*Next_sample_idx*" to read the coordinate's values of the next sample contour point. Then, the fetched coordinates are stored into the *sampledXcont* and *sampledYcont* register arrays. The contour sampler module stops when sample point number 128 is reached.

3.4 CVs Signature Computation Module

The proposed hardware implementation is intended as a hardware accelerator. In order to achieve lower time complexity, the number of mathematical operations needs to be reduced also the usage of the floating point numbers needs to be avoided during the design process. As discussed previously, the construction of the CVs signature matrix is based on creating many views for the shape contour from 128 viewing points sampled from a viewing circle. This circle is centered at the shape centroid. To do this, firstly a hardware that produces 128 sampled viewing points will have to be introduced. Then, a parallel architecture that performs the computation of the CVs signature matrix will be proposed.

3.4.1 Sampled Viewing Circle Points Computation Module

The objective of this module is to compute the coordinates of the viewing points lying on the viewing circle and store them into two register arrays (*VPXCoord* and *VPYCoord*). Figure 6 illustrates the hardware architecture of this module. As shown, the sampled viewing circle points computation module takes as input the output of the contour sampler module, which represents the coordinates of the 128 sampled contour points that were stored into two register arrays (*sampledXcont* and *sampledYcont*). This module comprises three main units: centroid coordinates computation unit, viewing circle radius computation unit, and a generator unit.

3.4.1.1 Centroid Coordinates Computation Unit

The original CVs signature algorithm computes the shape centroid coordinates as the average of the whole shape contour points coordinates. To simplify the computation of this step, the shape centroid computation is restricted to only the sampled contour points (128 samples instead of the whole contour points). The time needed to compute the shape centroid using the original CVs signature algorithm is a function of the number of contour pixels. However, by computing the shape centroid using only the sampled contour points, the processing time will be fixed (to perform 128 additions) regardless the number of contour pixels. Also, it eliminates the need for wide-ranging hardware divider, which reduces the hardware requirements considerably. The proposed hardware used to compute the shape centroid coordinates is very simple. Two adders are needed; the first one is used to compute sequentially the sum of all *x* coordinates of the 128 sampled contour points stored at the register array *sampledXcont*. The second adder is used to compute sequentially the sum of all *y* coordinates of the 128 sampled contour points stored at the register array *sampledYcont*. Since the coordinates of the 128 sampled contour points are integer values, an integer value will result for the sum of the coordinates of the sampled points. Then the average values of these two sums are computed by dividing them by 128. This division process can simply be implemented using a shifter to right shift these two sums by 7 bits. The shift logic is much simpler and faster than the division logic. Finally, two registers (x_c, y_c) are used to store the integer part of the result as the coordinates of the shape centroid.

3.4.1.2 Viewing Circle Radius Computation Unit

To compute the radius of the viewing circle, the city block distance between the shape centroid and all the sampled contour points is used instead of the Euclidean distance used by the original CVs signature algorithm. This simplifies the hardware implementation as well as avoiding the floating numbers that would have resulted from the square root needed by the Euclidean distance. The actual value of the viewing circle radius will be the maximum city block distance ($R_{\max}$) computed multiplied by 0.75. The factor 0.75 is chosen empirically, so that it achieves the highest recognition rate. To compute the city block distance, two comparators and two subtractors are used to get the absolute differences between the shape centroid and a sampled contour point (Δx, Δy). These two absolute differences are added together to get the city block distance. Each new city block distance computed is compared to a register named $R_{\max}$ (initialized to 0). If the new city block distance is larger than that stored in the register $R_{\max}$, the contents of $R_{\max}$ will be replaced by this new distance. In this way, $R_{\max}$ register will finally contain the maximum distance between the shape centroid and all sampled contour points. As illustrated previously, the radius of the viewing circle is $0.75*R_{\max}$. To multiply $R_{\max}$ by 0.75, a shifter is used to right shift the $R_{\max}$ value one bit to get $0.5 * R_{\max}$ then right shift $R_{\max}$ two bits to get $0.25 * R_{\max}$. Using an adder, the integer results of the two shift operations are added to get the viewing circle radius $R_{vc} = 0.75* R_{\max}$ as an integer value. In this way, the need for floating point multiplications to compute the factor $0.75* R_{\max}$ is eliminated.

3.4.1.3 Computing Sampled Viewing Circle Points Coordinates

Computing the coordinates of the sampled points of the viewing circle is a main challenge to implement in hardware, since they need heavy floating point calculations. So, to simplify the hardware implementation, an efficient method for computing the sampled viewing points coordinates without any floating point arithmetic's will be proposed. The coordinates of a sampled viewing circle point (x_{view}, y_{view}) can be computed by (1, 2) as follows:

$$x_{view} = x_c + R_{vc} \times \cos\theta \quad (1)$$

$$y_{view} = y_c - R_{vc} \times \sin\theta \quad (2)$$

Where, x_C, y_C are the coordinates of the shape centroid. The full period [Θ = 0: 2π] is sampled using 128 points. In the proposed hardware accelerator, if equations (1, 2) are implemented directly to compute 128 sampled viewing points, this will need 256 floating point multiplications. This unconditional hardware implementation causes an increase in both logic usage and the time complexity. The proposed hardware for computing the coordinates of the sampled viewing points depends on the fact that using only the first quarter values [0: π/2] of a sampled cosine function, the sampled cosine values for the other three quarters could be generated. By the same manner, the whole values of a sampled sine function could be generated using only the first quarter values of a sampled cosine function. This idea depends on that the order of the values of the sampled cosine and sine functions for each quarter can be considered either in the same order or in the reversed order as the values of the sampled cosine function for the first quarter. Also, the sign of these samples is taken into consideration. For example, the second quarter period of a sampled cosine function is in the reverse order of that of cosine function's first quarter after adding a negative sign. Another example regarding the sine function: the second quarter of a sampled sine function is in the same order as that of first quarter samples of a cosine function with the same sign. By the same idea, values of all quarters of sampled cosine and sine functions can be generated using only the first quarter values of a sampled cosine function. This means that only 32 values of the sampled cosine function for the first quarter need to be stored and from which the coordinates of the 128 sampled viewing circle points can be generated. These 32 values are stored in a lookup table as a 15 bits fixed point numbers, 1 bit for the integer part and 14 bits for the fraction. Then, a multiplier is used to perform only 32 multiplications sequentially to compute the term $R_{vc}*\cos\theta$ (in (1)), where θ is in the interval [0: $\pi/2$]. To simplify the multiplication process, the stored values of the sampled cosine function are considered as an unsigned integer binary numbers, and then they are multiplied by R_{vc}. The least 14 bits of the multiplication results (representing the fraction part) are ignored and the other bits are used to represent the result of the product term $R_{vc}*\cos\theta$. These results are stored in a 32*8 bits register array. As shown in figure 6, a generator is introduced that uses the stored register array values to generate the other values of the term $R_{vc}*\cos\theta$ for the interval [$\pi/2$: 2π].

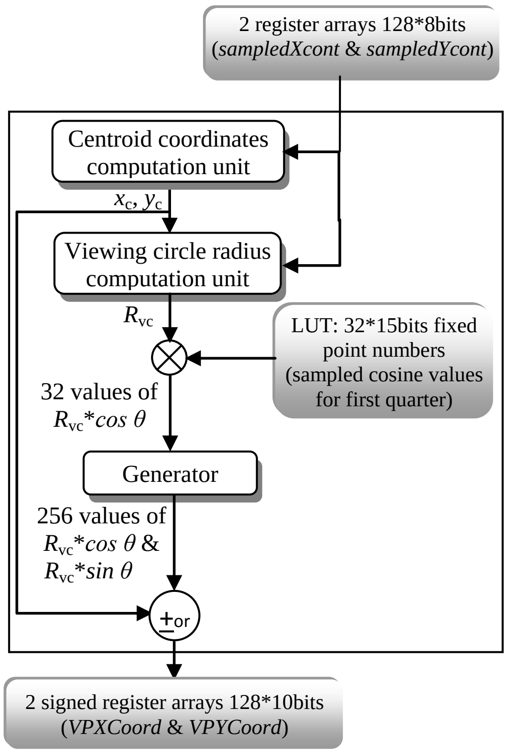

2 register arrays 128*8bits (*sampledXcont* & *sampledYcont*)
Centroid coordinates computation unit
x_c, y_c
Viewing circle radius computation unit
R_{vc}
LUT: 32*15bits fixed point numbers (sampled cosine values for first quarter)
32 values of $R_{vc}*\cos\theta$
Generator
256 values of $R_{vc}*\cos\theta$ & $R_{vc}*\sin\theta$
±or
2 signed register arrays 128*10bits (*VPXCoord* & *VPYCoord*)

Fig 6: The proposed hardware architecture of the viewing circle sample points computation module

Also, it generates the values of the term $R_{vc}*\sin\theta$ for the whole interval [0: 2π]. In both of these two cases there will be no need for any additional multiplications. To do this, the generator takes care by reversing the order of the 32 stored values for the factor $R_{vc}*\cos\theta$ as needed by the proposed method. Finally, according to (1, 2), to produce the coordinates of the 128 sampled viewing points, each output of the generator is added or subtracted (according to the expected sign for each generated value of the product terms $R_{vc}*\cos\theta$ and $R_{vc}*\sin\theta$) to or from the centroid coordinates (x_C, y_C).

As seen the proposed hardware for computing the viewing points coordinates reduces the multiplications needed to only 32 multiplications instead of 256 multiplications (12.5%) which also reduces the time complexity by the same factor. Also, the space needed to store the values of the sampled cosine and sine functions will be reduced to 12.5%. An important point is that the need for floating point multiplication is eliminated; merely, a simple binary integer multiplier is used. Finally, the coordinates of the 128 sampled viewing points are stored into two register arrays named *VPXCoord* and *VPYCoord*.

3.4.2 CVs Signature Matrix Computation Module

The main challenge faced when implementing the hardware accelerator of the CVs signature algorithm is the need of tremendous calculations since the size of the CVs signature matrix is 128*128. As seen, to compute the Euclidean distance between every sampled viewing point and all sampled contour points, 16384*2 subtractions, 16384*2 multiplications, 16384 additions, and 16384 square root operations are needed. To simplify the hardware implementation of the CVs computation, the squared Euclidean distance between every viewing point and all sampled contour points is used instead of the direct Euclidean distance. This simplification eliminates the need for additional hardware to compute the square root and as a result avoiding the need for floating point numbers. According to the nature of the CVs signature, each contour view from a certain viewing point on the viewing circle can be computed independently. Figure 7 illustrates the construction of the CVs signature matrix. As shown, it is obvious that each column represents one contour view from a certain viewing point VP_j and its computation is independent from the other columns (views). This will give a chance to use parallel architecture to compute the CVs signature matrix.

As shown in figure 8, the inputs of the CVs signature matrix computation module are two pairs of register arrays. One pair contains the coordinates of 128 sampled contour points (*sampledXcont* and *sampledYcont*) and the other contains the coordinates of 128 sampled viewing points (*VPXCoord* and *VPYCoord*). As illustrated in figure 8, CVs signature Matrix computation module comprises four processing elements that operate in parallel. Each processing element is composed of two subtractors, two multipliers and one adder to compute a squared Euclidean distance. Using four processing elements will cause a speedup on the CVs signature matrix computations. The columns of the CVs signature matrix are divided between the processing elements. So, each processing element will compute different 32 (128 views /4 processing elements) contour views from different 32 sampled viewing points. Here, 4 processing elements are used as an illustration. However, in the experimental results section, a study of the effect of changing the number of processing elements on the system speedup gained and hardware requirements will be made. Studying figure 7 carefully, it is obvious that each column (view) starts from a contour point whose index is the same as that of the viewing point VP_j. Hence, an indexing controller is designed that helps each processing element to start with the correct index when accessing the register arrays *sampledXcont* and *sampledYcont*.

The views that generated by the processing elements are stored in four block RAMs representing the CVs signature matrix. Ignoring the usage of the square root will lead to large data width of the block RAMs that stores the CVs signature matrix. According to the representations used for the coordinates of the sampled contour points and the sampled viewing points, the squared distance will need 21 bits binary number which is very large. In order to reduce the memory requirements for storing the CVs signature matrix, each squared distance is divided by the factor 1024 and the resulted integer parts are stored as the shape signature. This division is equivalent to right shift each element in the signature matrix by 10 bits which could be easily implemented on FPGA chip. As a result a memory is needed having only 11 bits data width

	VP_0	VP_1	VP_2	…	VP_{127}
0	*d* 0,0	*d*1,1	*d*2,2	…	*d*127,127
1	*d* 1,0	*d*2,1	*d*3,2	…	*d* 0,127
2	*d* 2,0	*d*3,1	*d*4,2	…	*d* 1,127
:	:	:	:	…	:
:	:	:	:		:
127	*d* 127,0	*d*0,1	*d*1,2	…	*d*126,127

Fig 7: Construction of the CVs signature matrix

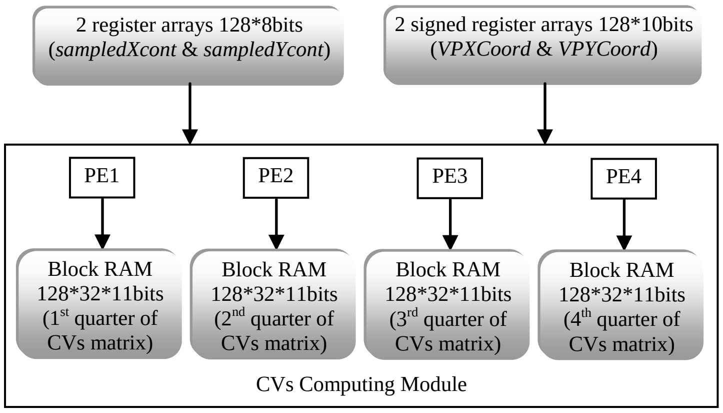

2 register arrays 128*8bits (*sampledXcont* & *sampledYcont*)
2 signed register arrays 128*10bits (*VPXCoord* & *VPYCoord*)
PE1
PE2
PE3
PE4
Block RAM 128*32*11bits (1st quarter of CVs matrix)
Block RAM 128*32*11bits (2nd quarter of CVs matrix)
Block RAM 128*32*11bits (3rd quarter of CVs matrix)
Block RAM 128*32*11bits (4th quarter of CVs matrix)
CVs Computing Module

Fig 8: Parallel architecture for computing CVs signature matrix

instead of 21 bits, which reduces memory space requirements significantly.

The original CVs signature algorithm performs row wise normalization for the CVs signature matrix. For row wise normalization, the squared distances in each row should be divided by the maximum value of the corresponding row. Division operations exhaust a large FPGA chip area and at least it takes number of clock pulses equal to the sum of the bit lengths of both dividend and divisor. This means that time and space costs needed will be very large using FPGA chips. So, the normalization process is postponed to the software part of the classification system.

4. EXPERIMENTAL RESULTS

In this work, the original CVs signature algorithm is implemented using Matlab on a PC having an Intel® core ™ i5-2430M CPU @ 2.4 GHz and 4 Giga bytes RAM. Only one viewing circle with a radius 0.9 of the maximum Euclidean distance between the shape centroid and the sampled contour points is used to compute the CVs signature. This software version of the original CVs signature algorithm is used for shape classification task. The MPEG-7 database is used for evaluation. Ten instances for each of the 70 classes composing MPEG-7 database are used for training 1NN classifier. The other 10 instances from each class are used for testing. The original CVs signature algorithm achieves 91.28% recognition rate.

As illustrated in section 3, the proposed modifications are introduced to the original CVs signature algorithm to reduce the processing time and the hardware requirements needed. To measure the robustness of the proposed modifications, a software version for the modified CVs signature algorithm is implemented using Matlab. Again, the MPEG-7 database is used for testing. The recognition rate for the modified CVs signature algorithm is measured the same way as used with the original CVs signature algorithm. The modified CVs signature algorithm achieves 89.85% recognition rate. As seen, the small reduction in performance using the proposed modifications can be ignored against the reduction in the hardware requirements and the computation time that will be achieved. This small reduction in performance proves the robustness of the modified CVs signature algorithm.

The proposed hardware accelerator for computing CVs signature is implemented using Verilog HDL language. The system is implemented on a Spartan-3 XC3S5000 FPGA with 74,880 logic cells, 1,872 kilo bits Block RAM, 784 user I/O pins [16]. The proposed hardware accelerator will be evaluated in terms of FPGA hardware requirements, and total computation time. Some binary images are chosen from the MPEG-7 database and are rescaled to 256*256 pixels to test the proposed hardware accelerator. The FPGA Hardware utilization is found to be 8.11% slice flip flops, 20.38% 4-input LUTs, 24.75% occupied slices, 2.88% 18x18 bits multipliers, and 40.38% block RAMs. The clock frequency is 29.51MHz. At this frequency, the computation time needed by the proposed hardware accelerator is found to be 0.673 milliseconds. The speedup achieved is 11.14 since the computation time needed by the software version of the original CVs signature algorithm (for the same image size) is 7.5 milliseconds. As seen, the proposed hardware accelerator achieves an acceptable FPGA chip utilization and a good speedup of more than 11 times.

The proposed hardware accelerator is re-implemented using more than one processing element (2, 4, 8 and 16 processing elements). Table 1 shows the computation time needed by the hardware accelerator against the FPGA hardware utilization for different number of processing elements. Using 16 processing elements a speedup of 50.34 times has been achieved. The choice of the number of processing elements is left to the specific application designer. So, a compromise between FPGA hardware requirements needed and speedup gained can be made.

Table 1. Effect of increasing no. of processing elements on speedup and the FPGA Hardware utilization

	FPGA HW Utilization					Available
	1 PE	2 PE	4 PE	8 PE	16 PE	
Slice Flip Flops	5404	5451	5546	5737	6120	66560
4 Input LUTs	13567	14802	17265	22191	32061	66560
Occupied Slices	8238	8874	10131	12644	17660	33280
MULT 18x18s	3	5	9	17	33	104
Block RAMs	42	42	44	48	48	104
Global Clocks	4	4	4	4	4	8
Computation time (milliseconds)	0.673	0.394	0.254	0.184	0.149	
Speedup	11.14	19.04	29.53	40.76	50.34	

5. CONCLUSIONS

The implementation of a hardware accelerator for feature extraction in binary images based on CVs signature algorithm is presented. In this work, some modifications to the original CVs signature algorithm are introduced. These modifications are intended to reduce time and space requirements for the proposed hardware accelerator. Robustness for the modified CVs signature algorithm is tested by incorporating it into 1NN classifier. Thus, a 1NN classifier software version based on the modified CVs signature algorithm is implemented using Matlab. The MPEG-7 database is used for evaluation. The modified version scores 89.85% recognition rate compared to 91.28% obtained by the original CVs signature algorithm. So, the modified CVs algorithm achieves a score that is too close to the original one while gaining a speed up of at least 11.14 (using only one processing element) and a significant reduction in hardware requirements.

The proposed hardware accelerator for computing CVs signature is implemented using Spartan-3 XC3S5000 FPGA. The system is designed using Verilog HDL language. One property of the CVs signature is that each contour view from a certain viewing point on the viewing circle can be computed independently. This gives us the opportunity to design a parallel hardware architecture for computing the CVs signature. Four versions for this parallel architecture are proposed. These versions use two, four, eight, and sixteen processing elements. A study relating the number of processing elements, speedup, and the FPGA hardware requirements needed has been done. The proposed hardware accelerator needs only 0.149 milliseconds (using 16 processing elements) for computing the CVs signature, while the corresponding software version for the original CVs signature algorithm needs 7.5 milliseconds on 2.4GHz core i5 processor with a 4 Giga bytes RAM. So, by using 16 processing elements, the proposed hardware accelerator achieves a speedup of 50.34. For the future work, an FFT IP core could be added to the proposed system to generate the CVs shape descriptor. In this way, the proposed hardware accelerator after adding the FFT IP core could be incorporated in a shape classification real time system. Also, the proposed system could be redesigned using pipelined architecture that speeds up computing the CVs shape descriptor.

6. REFERENCES

[1] M.A. Tahir, A. Bouridane, and F. Kurugoullu, “An FPGA based coprocessor for GLCM and Haralick texture features and their application in prostate cancer classification”, Analog Integrated Circuits and Signal Processing (2005) Vol. 43, Issue 2, 205–215.

[2] D. Nguyen, D. Halupka, P. Aarabi, and A. Sheikholeslami, “Realtime face detection, lip feature extraction using field programmable gate arrays”, IEEE Trans. Syst. Man Cybern. B Cybern. (2006) Vol. 36, Issue 4, 902–912.

[3] Nakano, T., Morie, T. and Iwata, A. 2003. A face/object recognition system using FPGA implementation of coarse region segmentation. In Proceedings of SICE Conference.

[4] Heikkinen, K. and Vuorimaa, P. 1999. Computation of two texture features in hardware. In Proceedings of the 10th International Conference on Image Analysis and Processing.

[5] Bariamis, D.G., Iakovidis, D.K., Maroulis, D.E. and Karkanis, S.A. 2004. An FPGA-based architecture for real time image feature extraction. In Proceedings of ICPR’04.

[6] Hedberg, H. Kristensen, F. and Owall, V. Implementation of a labeling algorithm based on contour tracing with feature extraction. IEEE International Symposium on Circuits and Systems.

[7] A.W. Malik, B. Thörnberg, M. Imran, and N. Lawal, “Hardware architecture for real-time computation of image component feature descriptors on a FPGA”, Hindawi Publishing Corporation, Int. J. Distributed Sensor Networks, Vol. 2014.

[8] F.F. Smach, J. Miteran M. Atri, J. Dubois , m. Abid, and J.-P. Gauthier, “An FPGA-based accelerator for Fourier Descriptors computing for color object recognition using SVM”, J. Real-Time Image Proc. (2007) vol. 2, 249–258.

[9] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object Recognition Using Shape Context”, IEEE Trans. Pattern Analysis and Machine Intelligence (2002) vol. 24, no. 4, 509-522.

[10] N. Alajlan, I.E. Rube, M.S. Kamel, and G. Freeman, “Shape retrieval using triangle area representation and dynamic space warping”, Pattern Recognition, (2007) vol. 40, 1911–1920.

[11] Felzenszwalb, P.F. and Schwartz, J. 2007. Hierarchical matching of deformable shapes. In Proceedings of IEEE CS Conference CVPR.

[12] H. Ling and D. Jacobs, "Shape classification using the inner-distance", IEEE Trans. Pattern Analysis and Machine Intelligence (Feb. 2007) vol. 29, no. 2, 286-299.

[13] H.D. Jomma and A.I. Hussein, "Circle views signature: a novel shape representation for shape recognition and retrieval", CJECE, to be published.

[14] Haig, T.D. and Attikiouzel, Y. 1989. An improved algorithm for border following of binary images. In Proceedings of IEE Eur. Conference Circuit Theory and Design.

[15] Ratnayake, K. and Amer, A. 2007. Sequential, irregular and complex object contour tracing on FPGA. In Proceedings of IEEE International conference ICIP.

[16] Xilinx Inc. Spartan-3 FPGA family data sheet. Available from www.xilinx.com, May 1997.