In the Development and Evaluation of Specialized Processors for Computing High-Order 2-D Image Moments in Real-Time

Nuno Roma
Nuno.Roma@inesc.pt

Leonel Sousa
las@inesc.pt

Instituto Superior Técnico / INESC-ID
Department of Electrical Engineering
Rua Alves Redol, No. 9 - 1000-029 Lisboa - PORTUGAL
Tel. +351 21 3100000 - Fax: +351 21 3145843

Abstract

Image moments are used in image analysis for object modelling, matching and representation. The computation of high-order moments is a computational intensive task that can not be implemented in real-time with nowadays general-purpose processors. This paper proposes a set of specialised processors for generating an image moment of an arbitrary order in real time, by adopting systolic processing techniques and floating-point arithmetic units. It proposes a modular and cost effective architecture for generating image moments, with a processing time not dependent on the order of the computed moments. The architecture was implemented using different devices, such as programmable digital processors, configurable hardware logic and integrated circuits, by using a $0.7\ \mu m$ *CMOS process. The several implementations have shown the effectiveness of the architecture, and the obtained results allow us to compare the different solutions in terms of speed, flexibility, cost and power consumption.*

Keywords: High-order image moments, Pipeline architectures, Systolic processing, Digital Signal Processors

1. Introduction

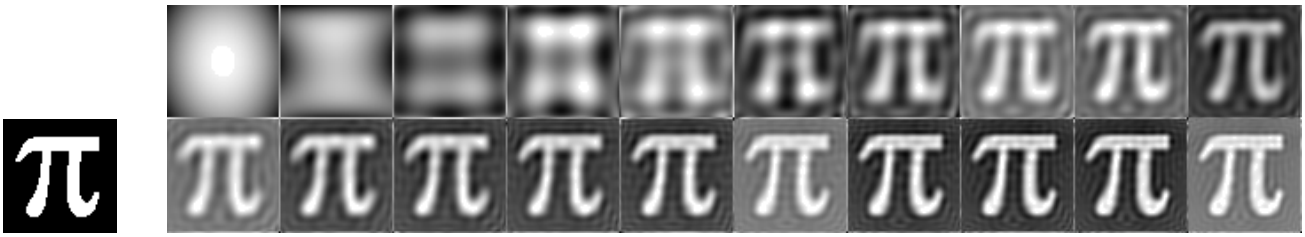
Two-dimensional object representation and recognition is an important topic in the computer vision area. Moments of the intensity function of pixels are commonly used for representing an object or an image. The $p = m + n$ order moment $(\mathcal{M}_{m,n})$ of an array of pixels with values $f(x, y)$ and size NM is defined by equation 1 [1, 2].

$$\mathcal{M}_{m,n} = \sum_{x=1}^{N} \sum_{y=1}^{M} x^m y^n f(x, y) \qquad (1)$$

Low order moments are commonly used to extract features, such as area and center of gravity, and to find the location and orientation of objects in a image [1, 2, 3, 4]. In contrast, high order moments are used for pattern recognition and image representation [5, 6].

A direct computation of equation 1 requires NM additions and $(m + n)NM$ multiplications. In the last years, several algorithms and architectures have been proposed to improve the speed of moments computation. However, some of them are only valid for low-order moments and/or for binary images [7, 8, 9, 10]. Other proposals suggest architectures without practical interest due to its complexity [11]. Moreover, floating-point arithmetic has to be used to compute moments of different orders, given that the required dynamic range increases very rapidly with the moment's order. Some other proposed architectures [7] adopt wavefront processing techniques to adapt the throughput to the variable processing time exhibited by floating-point arithmetic units. However, although they eliminate the need to respect the "worst" processing period, they require additional circuits to exchange data between Processing Elements (PE), which represents a significant overhead in the hardware cost.

This paper proposes a new architecture for computing 2-D gray level image moments with any order in real-time, by adopting systolic processing techniques and floating point arithmetic. The proposed architecture is modular and scalable for higher order moments and requires a reduced amount of hardware. Moreover, the results presented in this paper demonstrate that the architecture is well suited to implement specialised processors for computing image moments based on Digital Signal Processors (DSP), Field Programmable Date Arrays (FPGA) and Application Specific Integrated Circuits (ASIC). By comparing the different implementations, it is possible to conclude that the real-time processing is always achieved. However, some advantages



(a) (b)

Figure 2. Representation and reconstruction of an image pattern using high order moments: (a) Original input image. (b) Reconstructed image with up to second-order moment through up to 40th-order moment, from top to bottom rows and from left to right.

of the specific characteristics of each of the considered processor implementations can be taken, in what concerns the operating speed, flexibility and power consumption.

This paper is organised as follows. Section 2 presents two main application examples of the use of image moments in image processing. An efficient *power core* for generating the x and y powers is proposed in section 3. In section 4 a new systolic architecture is proposed and in section 5 floating point arithmetic units are described. Section 6 presents and compares several implementations of a proposed processor based on different devices, such as: FPGAs (section 6.1), ASICs (section 6.2) and DSPs (section 6.3). Section 7 concludes the presentation.

2. 2D Image Moments

As it has been referred in the previous section, image moments are often used in computer vision to provide a set of image features for different applications, such as localization, pattern recognition and image representation [1, 2]. The zero through second order moments provide information about the area, center of mass and about the approximation of an object to an ellipse [3] (see figure 1). Moments with order lower than $p = 6$ can also be combined to form moment invariants to certain shape transformations, such as translation, rotation and scaling [4].

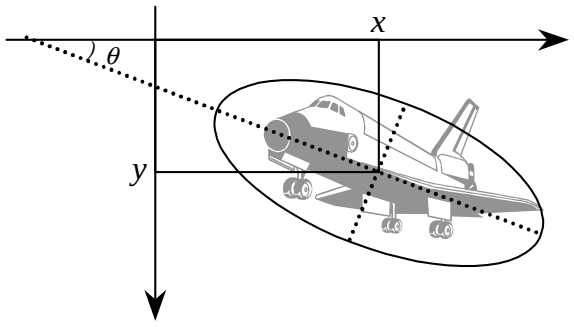


Figure 1. Definition of several localization parameters through the computation of the image ellipse.

Besides feature extraction, a significant research effort has recently been made to use high order image moments in pattern recognition and image representation applications [5, 6]. In fact, it has been shown that high order moments can provide a means of reconstructing images or patterns using a finite set of moments. This makes it possible to use fewer bytes to represent the reconstructed image than the required to represent the original image. However, since the usage of an infinite number of image moments is not feasible, this representation presents always an inherent error that is dependent on the maximum order of the set of used moments (see figure 2).

3. Power Core

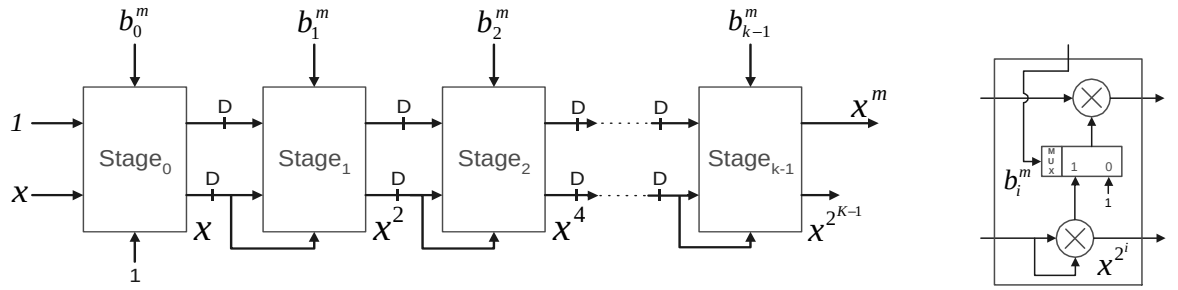
To compute the values of the x^m and y^n powers with a traditional sequential architecture, m and n clock cycles are required, respectively. Such basic architecture leads to a great amount of computation time for high order moments. According to [7], a faster architecture can be obtained by applying the following decomposition:

$$m = \sum_{i=0}^{k-1} b_i^m 2^i \quad , \quad n = \sum_{i=0}^{k-1} b_i^n 2^i \tag{2}$$
$$k = \lceil \max \{\log_2 (m+1), \log_2 (n+1)\} \rceil$$

Hence, the expressions for the powers x^m and y^n take the following form:

$$x^m = x^{b_0^m . 2^0 + b_1^m . 2^1 + \ldots + b_{k-1}^m . 2^{k-1}} = \prod_{i=0}^{k-1} x^{b_i^m . 2^i} \tag{3}$$

$$y^n = y^{b_0^n . 2^0 + b_1^n . 2^1 + \ldots + b_{k-1}^n . 2^{k-1}} = \prod_{i=0}^{k-1} y^{b_i^n . 2^i} \tag{4}$$



(a) Architecture **(b) PE**

Figure 3. Power-Core module.

Thus, it is possible to conclude that, in order to compute the value of x^m, it is only necessary to perform $2k$ multiplications: k of them for computing the x^{2^i} terms and the remaining k to calculate the product of the $x^{b_i^m \cdot 2^i}$ terms. Figure 3 represents a structure with k stages to compute equation 3. If the input values were composed by the sequence $\{1, 2, 3, \ldots, N\}$, the sequence $\{1^m, 2^m, 3^m, \ldots, N^m\}$ would be obtained at the output. In this systolic architecture, the overall structure is operated in a synchronous way, with the D symbol representing a delay of one clock cycle. With this structure, it is possible to obtain one output value in each clock cycle, with a latency of k clock cycles.

4. Dedicated Pipeline Architecture for Computing an Image Moment

The computation of $\mathcal{M}_{m,n}$ given by equation 1 can be represented by a vector-matrix product followed by a vector dot product, as depicted in equation 5. In these equations, F represents the $N \times M$ image matrix, while X and Y represent the N and M sized vectors composed by the set of x^m and y^n powers, respectively. According to [7], the computation of this equation can be represented by the planar Dependence Graph (DG) presented in figure 4, corresponding to an array architecture composed by $N \times (M+1)$ PEs.

$$\begin{aligned} \mathcal{M}_{m,n} &= XFY = XG = HY \qquad (5) \\ X &= \left[\begin{array}{cccc} 1^m & 2^m & \cdots & N^m \end{array}\right] \\ Y &= \left[\begin{array}{cccc} 1^n & 2^n & \cdots & M^n \end{array}\right] \\ F(x,y) &= \{f(x,y) : x = 1 \ldots N, y = 1 \ldots M\} \end{aligned}$$

For real images, with a great number of pixels, it is convenient to reduce the dimension of this array. This can be achieved by projecting the DG on a linear structure with a valid pair of *projection* and *schedule* vectors $(\vec{d}, \vec{s})$. The systolic structure presented in figure 5 is obtained by defining

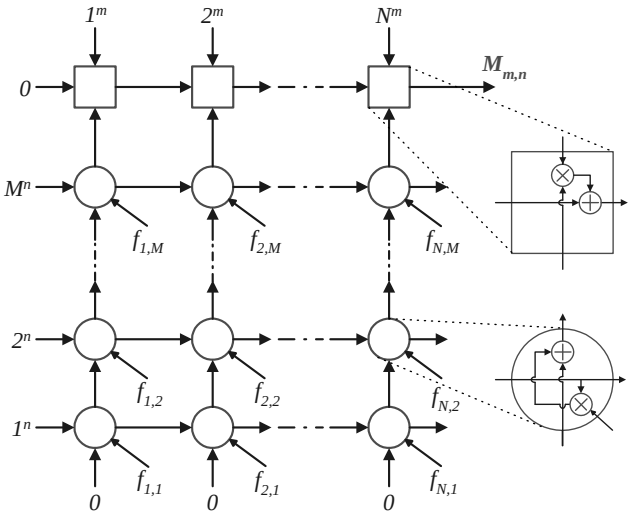


Figure 4. 2-D systolic architecture derivation: Dependence Graph (DG).

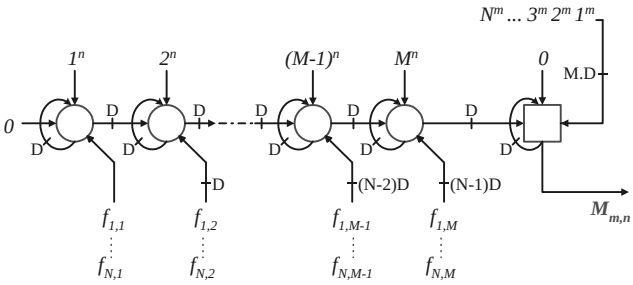


Figure 5. 2-D systolic architecture derivation: Linear Signal Flow Graph (SFG).

$\vec{d} = [\,1\ \ 0\,]^T$ and $\vec{s} = [\,1\ \ 1\,]^T$, while maintaining the same type of PEs.

Although the dimension of the processor has been reduced, this structure still requires a large amount of hardware, namely, for images with median and high resolutions (e.g. N=M=512). According to [7], this problem can

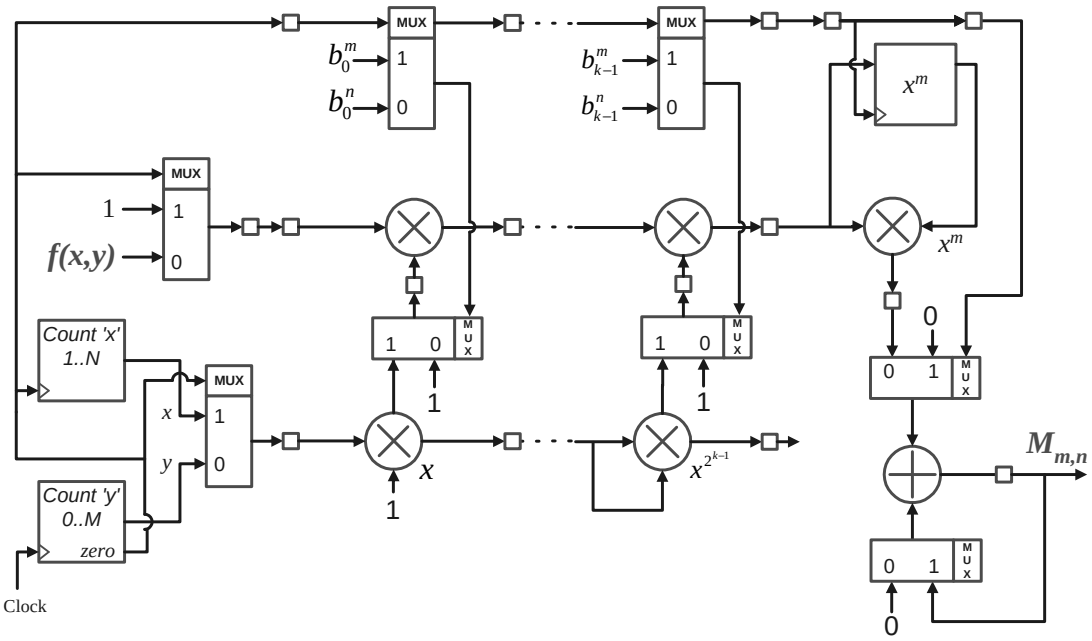


Figure 7. Block diagram of the serial processor.

be solved by regularly partitioning the Signal Flow Graph (SFG) into blocks, each consisting of a cluster of PEs [12]. Figure 6 presents the limit situation, where the set of all PEs represented by circles is mapped on a single PE, leading to a serial pipeline architecture. This configuration represents the most economical solution in what concerns hardware resources. In fact, only one multiplier is used in the PE represented by a circle and partial results are accumulated in the output PE, as depicted in figure 6.

A complete diagram of the described serial systolic processor for the computation of an image moment with any order is shown in figure 7, with the square symbol ($\square$) representing a processing delay. In order to use a single power core to compute the powers of x^m and y^n, a register was used in the final stage of the processor to store the x^m value of the line being processed. Furthermore, in order

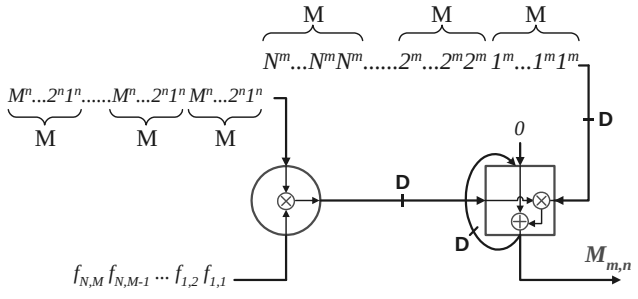


Figure 6. 2-D systolic architecture derivation: Serial pipeline structure.

to increase the processing frequency of the power-core, a pipeline register was introduced between the multiplier and the adder, thus reducing the overall critical path.

The serial pipeline processor shown in figure 7 uses a total of $2k+1$ multipliers and 1 adder. The total processing time required to calculate the $\mathcal{M}_{m,n}$ moment of an $N \times M$ image is:

$$T = [N\,(M+1) + t_{pow} + 2] \times t_{op}, \tag{6}$$

where t_{pow} is the latency of the power-core circuit given by:

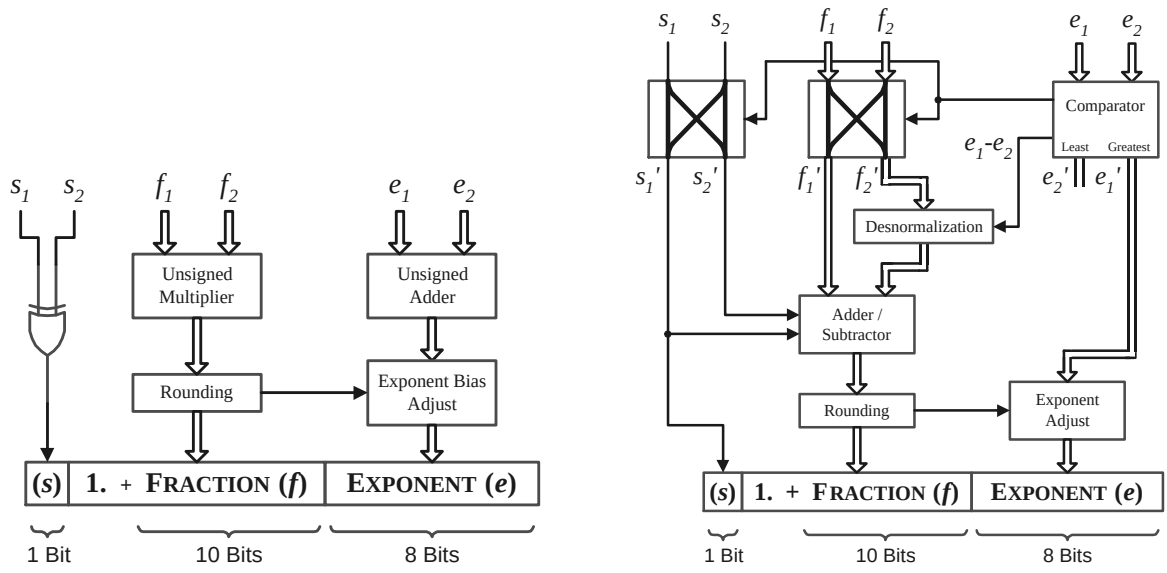
$$t_{pow} = 1 + \log_2\left(\lceil \max\{m+1, n+1\}\rceil\right) + 1 \tag{7}$$

and t_{op} is the maximum time required to perform a single floating-point multiply or addition operation. Hence, the total processing time is:

$$T = [N \times (M+1) + \log_2\left(\lceil \max\{m+1, n+1\}\rceil\right) + 4] \cdot t_{op} \tag{8}$$

5. Arithmetic Units

Due to the great number of arithmetic operations involved in the computation of the image moments, some of the most critical blocks of the proposed processor are the arithmetic units. In fact, its overall performance is remarkably dependent on the efficiency of these elements. As it was referred in the previous sections, floating-point arithmetic units are used to accommodate high-dynamic ranges required to compute high-order image moments with smaller word widths. Experimental results show that the



(a) Multiplier **(b) Adder**

Figure 9. Floating-point arithmetic unit.

used floating-point precision, whose formats is presented in figure 8, seems to be enough (1 sign bit (s), 8 bits for the exponent (e) and 10 bits used for the fractional part (f) of the mantissa). In fact, in the present application, the sign bit could be eliminated, since image data is usually represented with unsigned values. However, it was decided to implement the arithmetic units with a signed representation, in order to extend its application to different sources of data.

The exponent is represented as a biased signed number, with a bias of 127. The mantissa is composed by a hidden bit, equal to 1, and by a fractional part, which represents a number smaller than 1. Hence, the floating-point number is represented by the value of $(-1)^s \times \left[1.f \times 2^{e-127}\right]$.

5.1. Floating-Point Multiplier

The floating-point multiplication is generally computed according to equation 9 (where $M_i = 1.f_i$ and $E_i = e_i - 127$):

$$\left[(-1)^{S_1} \times M_1 \times 2^{E_1}\right] \cdot \left[(-1)^{S_2} \times M_2 \times 2^{E_2}\right] = \quad (9)$$

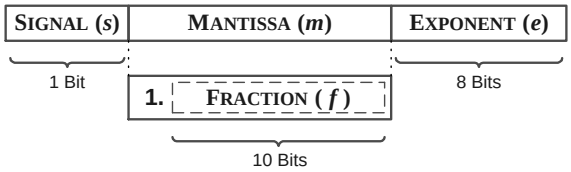


Figure 8. Floating-Point Format.

$$= (-1)^{S_2 \oplus S_2} \times (M_1 \cdot M_2) \times 2^{(E_1+E_2)}$$

The result of this operation is thus obtained by making use of an unsigned multiplier to compute the mantissa of the result, an unsigned adder to compute the exponent, and a barrel shifter to accomplish the rounding of the fractional part of the product, in order to normalise the result. Hence, this operation is performed in 4 steps, as illustrated in figure 9(a):

i) the mantissa of the operands are multiplied;

ii) the result is rounded in order to obtain a normalised 10-bit number;

iii) the new exponent is calculated by adding e_1 and e_2 and by taking in consideration the previous rounding step;

iv) since the exponents are stored with a biased format, the bias constant is subtracted from the sum of the biased exponents:
$(e_1 - 127) + (e_2 - 127) + 127 = e_1 + e_2 - 127.$

The last step can be done simultaneously with the third step. The sign value is simply the result of the XOR operation $(\oplus)$ between s_1 and s_2.

5.2. Floating-Point Adder

The floating-point addition is somewhat more complex than the multiplication because, depending on the signs of the operands, it may actually be a subtraction. Furthermore, it is necessary to start by adjusting the operands, so that they

have the same exponent when they are applied to the adder, according to the following formula, where $e_1 \geq e_2$:

$$(f_1 \times 2^{e_1}) + (f_2 \times 2^{e_2}) = \left[f_1 + f_2 \times 2^{-(e_1 - e_2)}\right] \times 2^{e_1} \quad (10)$$

This operation is performed in 5 steps, as illustrated in figure 9(b):

i) the exponents are compared and the operands are swapped so that $e_1 - e_2 \geq 0$, and the exponent of the result is temporary assumed to be e_1;

ii) the mantissa of the smaller operand is right shifted $d = e_1 - e_2$ places by using a barrel shifter;

iii) the result of this shift and the mantissa of the greater operand are applied to the adder/subtracter, taking in consideration the signs of both operands;

iv) the result of this operation is rounded and shifted until it is normalised and the exponent of the result is adjusted;

v) the sign of the result is determined.

The last step can be done simultaneously with any of the previous steps.

5.3. Floating-Point Units Implementation

The floating point units were described using synthesizable VHDL code. A standard library of arithmetic units was used to implement the adders and multipliers [13]. This library contains many units for a comprehensive set of integer arithmetic operations, with multiple structurally different implementations, allowing the adjustment of the performance of the circuit by trading circuit area versus speed. Among the several available arithmetic units, fast multipliers and adders, implemented using the carry-save technique, were selected. Furthermore, Wallace trees were adopted to speed up carry-save addition in binary adders and multipliers.

6. System Implementation

The choice of the hardware support to be used in the implementation of the described architecture is a significant aspect in what concerns the final characteristics of the system. Three main aspects are usually considered:

- *System Performance*, which characterises the operating frequency of the system, as well as its power requirements.
- *System Configurability*, characterising the processor modularity according to a set of implementation parameters, as well as its programmability.
- *Implementation Costs* of the system.

In order to obtain a detailed comparison of the characteristics of several different implementation supports, three distinct approaches were considered: Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC) and Digital Signal Processor (DSP). Each of these hardware implementations can be distinctly characterised in what concerns the three main aspects referred before. Application specific integrated circuits are often regarded as being the obvious choice, offering better conditions to provide the necessary signal processing performance required by a real-time operation. However, such dedicated hardware has the disadvantage of lacking flexibility and modularity, making it difficult to introduce later modifications on the algorithm or on its parameters. In contrast, DSP based implementations offer a programmability capability which usually makes them the optimal choice in what concerns system flexibility. However, applications with real-time demands usually require signal processing performances which cannot be provided by standard DSPs. Such applications are often implemented by using dedicated co-processors to perform the more computational intense tasks. FPGAs can be considered as a compromise between these two options. Although these configurable devices have a hardware structure more closed to the ASIC's structure, they offer a programming flexibility comparable to the one offered by DSPs.

In the following subsections it will be described and compared the several implementations of the proposed architecture. In what concerns the system programmability and parameterisation, the three referred implementations can be classified into two categories: *hardware implementations*, where VHDL hardware description language has been used in the architecture description of the FPGAs and of the ASIC, and *software implementations*, where Assembly programming language has been used to program the DSP.

6.1. FPGA based implementation

A modular processor for generating high-order image moments, based on the architecture presented in the previous sections, was physically implemented in a small number of low-cost FPGAs. Comparing figure 10 with figure 7, it can be observed that an additional pipeline register was introduced in the accumulator circuit, in order to balance the processing time of the pipeline stages. With this additional register, the odd and even partial results are independently accumulated in two registers. At the end of the image, the adder is used to compute the final result.

The processor is composed by three different types of blocks. Module A' generates x and y values and introduces, at the correct time, the pixel intensity values into the power-core. Each module of type B implements one stage of the power-core. Module A'' receives, as input, the several val-

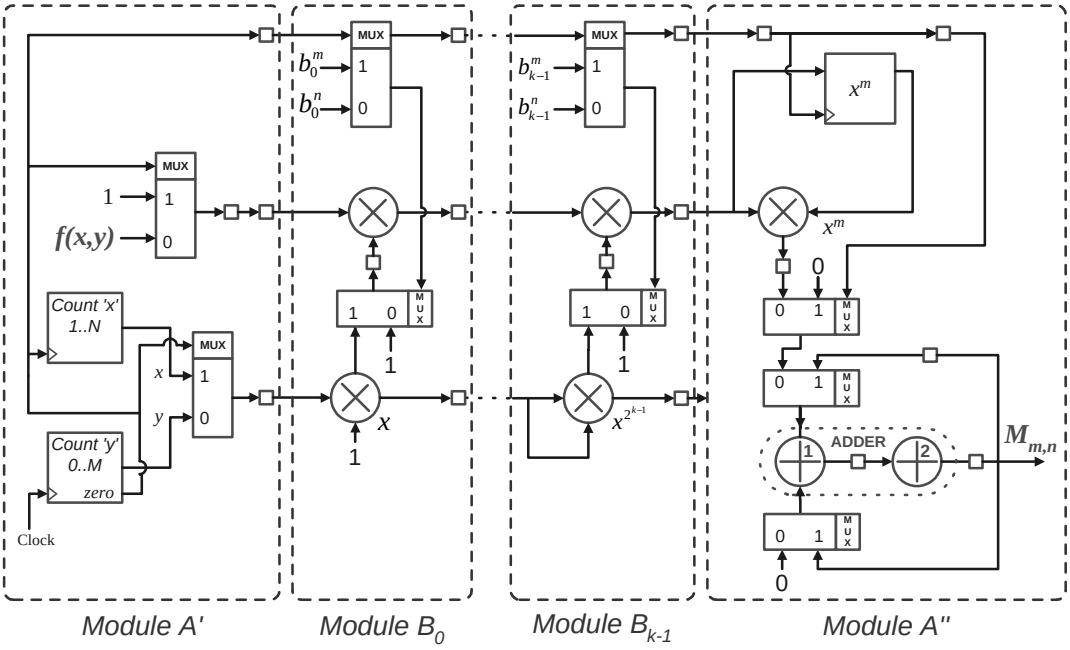


Figure 10. Processor implementation with Xilinx XC4013E FPGAs.

ues of x^m (which are sequentially stored in a register) and the pre-calculated values of $y^n \times f(x,y)$. The moment value is computed by multiplying these values along the set of N image lines: $\mathcal{M}_{m,n} = \sum_{x=1}^{N} \sum_{y=1}^{M} x^m \left(y^n f(x,y)\right)$. A single module of type A' and A'' and a set of k ($k = \lceil \max\{\log_2(m+1), \log_2(n+1)\} \rceil$) modules of type B are used to compute the 2-D image moments $\mathcal{M}_{m,n}$. As an example, only three or four modules of type B are required to compute moments of any order $p = (m+n) \leq 14$ (with $m, n \leq 7$) or $p \leq 30$ (with $m, n \leq 15$), respectively.

The processor modules were implemented using the XC4013E-PQ160 FPGA [14] and the Xilinx Foundation Series 2.1 software [15]. This configurable device is one of the most basic FPGAs of the XC4000E family, composed by 576 Configurable Logic Blocks (CLBs) and 129 I/O pads [14]. Module A' requires little hardware: just two counters and one multiplexer. For this reason, it was implemented together with module A'' ($A = A' + A''$) in a single FPGA (see figure 10). In table 1 it is presented the results of these implementations, regarding the device occupation and the maximum working frequency of the processor. The obtained results evidence that both types of modules A and B can be implemented in a single XC4013E-PQ160 FPGA. They occupy about 78 % of the available set of CLBs and about 98 % of the available I/O pads. In figure 11 it is presented the relation between the computed moment order (p) and the required number of XC4013E-PQ160 FPGAs. The maximum working frequency achieved is about 10 MHz, which makes the processor able to compute 2-D image moments at the video-rate required for resolutions of 640×480 pixels, as it is depicted in figure 11. The value of the work-

Table 1. Implementation results of the several modules that compose the FPGA based processor.

Module	Occupation [%]	Minimum Period [ns]	Frequency [MHz]
$A = A' + A''$	78.3	99.5	10.1
B	75.3	69.4	

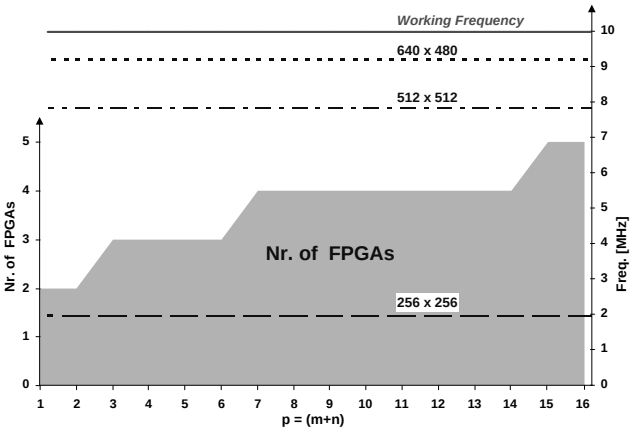


Figure 11. Number of required FPGAs to compute a p=(m+n) order moment (with m,n $\leq$ 7); required clock frequency to process several image sizes.

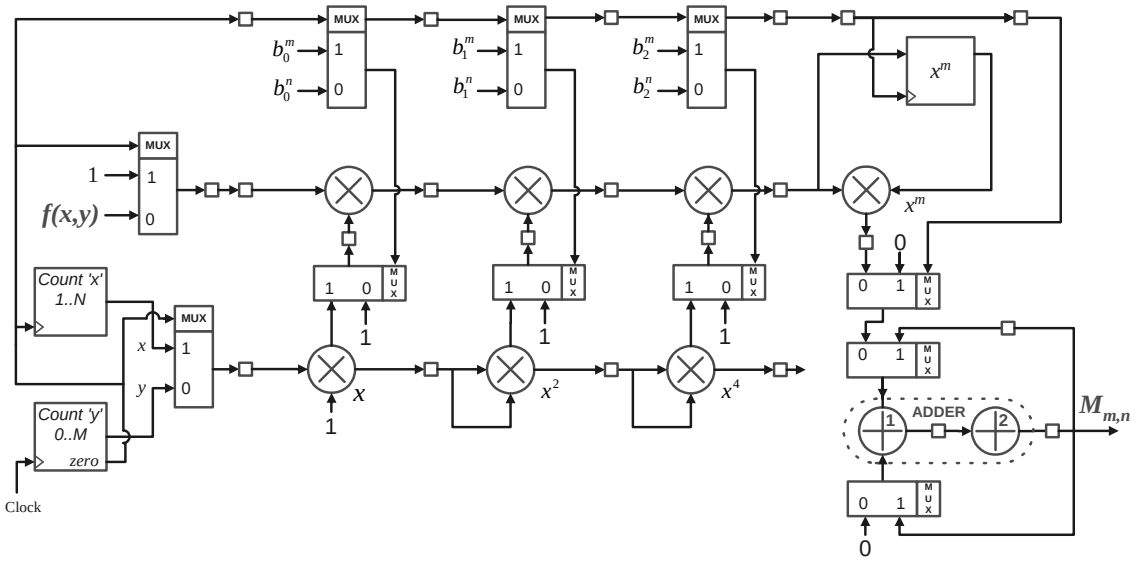


Figure 12. Processor implementation using an ASIC device.

Table 2. FPGA based processor implementation results.

Parameter	Value	
Devices Used	4	$FPGAs$
Logic Blocks	1319	$CLBs$
Power Dissip.	4.67	W
Critical Path	99.47	ns
Proc. Time	26.13	ms

ing frequency can still be increased by reducing the processing time of the arithmetic units, which can be achieved by increasing the pipeline depth.

A processor composed by four XC4013E-PQ160 devices, for generating image moments with order less than 14 (with $m, n \leq 7$), was implemented. In table 2 it is depicted its electrical characteristics, where the presented processing time refers to the computation of a 512×512 image moment. This processor was connected and fully tested with an acquisition video processing system, which accepts PAL video signals and sends the computed image moments to a Pentium Personal Computer, through one I/O parallel port.

6.2. ASIC based implementation

The system implementation supported on the hardware structure of an ASIC device is based on the architecture previously described in section 4 and presented in figure 12. Similarly to the FPGA based processor, this implementation has been carried out to obtain an efficient hardware processor capable of computing image moments with order less than 14 (with $m, n \leq 7$). As it has been done with the previously described FPGA based device, an additional pipeline register was introduced in the accumulator circuit in order to increase the working frequency of the processor. Likewise, the parameterisation of the order of the computed image moment is done simply by programming two processor registers.

The VHDL description of the circuit was synthesised with Synopsys^{TM} Design Compiler. It was implemented using the Atmel ES2 Standard Cells Library for the $0.7\ \mu m$ dual-metal CMOS process, based on a $5.0\ V$ technology [16]. This library is composed by a complete set of StdLib logic cells, whose simplest device, the inverter, is characterised by a typical propagation time of about $0.12\ ns$, occupying a total area of $273.6\ \mu m^2$.

The implementation results of the synthesised circuit are presented in table 3. By comparing these results with those obtained for the FPGA based processor (see table 2), it can be observed a reduction of about 88.5% in power consumption. Moreover, the working frequency can be increased to $23.36\ MHz$, which decreases the processing time by a factor of 2.32.

Table 3. ASIC implementation results.

Parameter	Value	
Critical Path	42.80	ns
Cells	4716	
Transistors	43894	
Area	3.26	mm^2
Power	534.4	mW
Proc. Time	11.242	ms

```
; x^m Power Computation (result in R0)

      ldf    0,R1        ; R1 <- x (line counter)
      ldi    @_tableX,AR0
      ldi    @powerX,R7  ; R7 <- m
L1:   ldi    511,RC      ; Repeats the following
                         ; block 511+1 times
      rptb   L2

Lx1:  lsh    -1,R7       ; Cflag <- b0
      bncd   Lx2         ; Tests bo bit
      ldi    1,R0        ; R0 <- 1
      mpyf3  R1,R1,R2    ; R2 <- x^2
      mpyf3  R2,R2,R3    ; R3 <- x^4

      mpyf   R1,R0       ; R0*=b0.x
Lx2:  lsh    -1,R7       ; Cflag <- b1
      bncd   Lx4         ; Tests b1 bit
      addi   1,R1        ; R1++ (for next cycle)
      nop
      nop

      mpyf   R2,R0       ; R0*=b1.x^2
Lx4:  lsh    -1,R7       ; Cflag <- b1
      bnc    Lx4         ; Tests b2 bit

      mpyf   R3,R0       ; R0*=b2.x^4
L2:   stf    R0,*AR0++   ; tableX[AR0++] <- x^m
```

(a)

```
; Moment Mmn computation
;  - tableX contains the x^m power values
;  - tableY contains the x^n power values
;  - frame contains the pixel value at line x and column y

      ldi    @_tableY,AR0
      ldi    @_tableX,AR1
      ldi    @_frame,AR7
      ldi    0,R2
      ldi    0,R3
      ldi    512,R6          ; repeats next block 512 times

      subi   1,R6            ; counter --
L7:   mpyf   *AR7,*AR0++,R0  ; R0 <- frame * tableY[counterY++]
      rpts   510             ; repeats next block 511 times
      mpyf3  *AR7,*AR0++,R0  ; R0 <- frame * tableY[counterY++]
   || addf   R0,R2           ; R2 += R0

      addf   R0,R2           ; R2 += R0
      mpyf3  *AR1++,R2,R1    ; R1 <- tableX[counterX++]*R2
      addf   R1,R3           ; R3 += R1

      cmpi   0,R6
      bnzd   L7
      ldi    @_tableY,AR0
      ldi    0,R2
      subi   1,R6            ; R6--
                             ; R3 = Moment Value
```

(b)

Figure 13. Assembly code: (a)-Initialisation of the table containing the x^m values; (b)-moment computation.

6.3. DSP based implementation

In order to compare the previously described dedicated *hardware implementations* with an implementation based on a digital signal processor, a *software implementation* has been performed using a standard floating-point DSP. The chosen device was the 32-bit DSP TMS320C30 from Texas Instruments, based on a $1.0\ \mu m$ CMOS technology and operated using a clock frequency of $33\ MHz$. This processor is characterised by having a $60\ ns$ single-cycle instruction execution time, capable of performing 33.3 MFLOPS and 16.7 MIPS [17]. This DSP exhibits greater precision than the hardware implementations previously described (the mantissa is represented with 24 bits).

In order to obtain an algorithm as efficient as possible, the computation of the values of x^m and y^n was done by making use of two distinct tables, pre-computed during an initialisation phase. This, of course, has the inherent disadvantage of loosing part of the flexibility of the proposed architecture. Nevertheless, due to the intrinsic sequential execution characteristics of the DSP (contrasting with the parallel structures of the FPGAs and ASIC), this is the most efficient method to compute these power values. Moreover, the other alternative would require the repetitive computation of the same set of values for all processed frames. Using this method, these tables are computed only once, during the initialisation phase of the processor.

Therefore, two distinct main tasks must be considered in this implementation: the initialisation of the x^m and y^n tables and the computation of the image moment. The Assembly code of these tasks (correspondent to processing of a (512×512) image) is depicted in figure 13(a) and in figure 13(b), respectively.

A total of $((512 \times 19) + 8)$ cycles are required to initialise the tables containing the x^m and y^n power values (with $m, n \leq 7$). Therefore, the initialisation task correspondent to the pre-computation of the x^m and y^n tables comprises about 19472 execution cycles, which corresponds to a initialization time of about $1.17\ ms$. In what concerns the computation of the moment value, it is possible to conclude that $(524 \times 512 + 7)$ execution cycles are required, which corresponds to a processing time of about $16.1\ ms$. Table 4 depictes the main characteristics of this DSP based implementation. It can be observed that while the processing time value is between those obtained for the other two implementations, the obtained power consumption value presents an increase by a factor of about 2.33 when compared with the ASIC based implementation. With this processor, moments with order $p \leq 14$ ($m, n \leq 7$) can be computed for 512×512 images at a rate of about 60 frames per second.

Table 4. DSP based implementation.

Parameter	Value	
Devices Used	1	DSP
Power Dissip.	1.25	W
Clock Frequency	33.0	MHz
Execution Cycle	60	ns
Processing Time	16.1	ms

7. Discussion and Conclusions

This paper proposes a new systolic architecture for computing high order 2-D gray level image moments, $\mathcal{M}_{m,n}$, requiring only $(2k+1)$ multipliers $(k = \lceil \max\{\log_2(m+1), \log_2(n+1)\}\rceil)$ and one adder. The described architecture makes use of floating-point representation to accommodate the high dynamic range of the operands with a small number of bits. As far as we know, this is the first architecture that allows the design of real-time processors for computing 2-D image moments with a number of multipliers that only increases with the logarithm of the maximum order of the moments in one dimension. Moreover, the repetitive structure of the architecture is well adapted to the development of modular processors with VLSI circuits and with configurable devices.

It has been shown the usefulness of the proposed architecture for developing real-time processors based on the different type of devices: FPGAs, ASICs and DSPs. The ASIC based implementation has proved to be the most suitable to be used in applications requiring low-power consumption, such as mobile equipment. It has also been shown that DSP based implementations are more suitable for applications where the proposed architecture is used as a dedicated co-processor in a host processing system and where power consumption is not a significant concern. The results of the FPGA implementation have shown that this configuration is well suitable only in the design and test phases of the dedicated processors, due to the high power consumption and moderate operating frequency.

References

[1] M.-K. Hu, "Visual Pattern Recognition by Moment Invariants", *IRE Transactions in Information Theory*, vol. IT-1, February 1962, pp. 179–187.

[2] R. J. Prokop and A. P. Reeves, "A Survey of Moment-Based Techniques for Unoccluded Object Representation and Recognition", *Graphical Models and Image Processing*, vol. 54, no. 5, September 1992, pp. 438–460.

[3] G. Agin, "Handbook of Industrial Robotics", chap. *Vision Systems, Addison-Wesley*, 1985.

[4] R. Haralick and L. Shapiro, "Computer and Robot Vision", *Addison-Wesley*, vol. II, chap. 18. 1993.

[5] C.-H. Teh and R. T. Chin, "On Image Analysis by the Methods of Moments", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, July 1988, pp. 496–513.

[6] S. X. Liao and M. Pawlak, "On Image Analysis by Moments", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, March 1996, pp. 254–266.

[7] C. Coelho, N. Roma, L. Sousa, "Pipeline Architectures for Computing 2-D Image Moments", *Proc. of DCIS'99*, Palma de Mallorca, Spain, November 1999.

[8] R. Andersson, "Real-Time Gray-Scale Video Processing Using a Moment-Generating Chip", *IEEE Journal of robotics and Automation*, vol. 1, no. 2, June 1985, pp. 79-85.

[9] K. Chen, "Efficient Parallel Algorithms for the Computation of Two-Dimensional Image Moments", *Pattern Recognition*, vol. 23, no. 1/2, 1990, pp. 109-119.

[10] X. Jiang and H. Bunke, "Simple and Fast Computation of Moments", *Pattern Recognition*, vol. 24, no. 8, 1991, pp. 801-806.

[11] H. D. Cheng, C. Y. Wu and D. L. Hung, "VLSI for Moment Computation and its Application to Breast Cancer Detection", *Pattern Recognition*, vol. 31, no. 9, 1998, pp. 1391-1406.

[12] P. Pirsch, "Architectures for Digital Signal Processing", *John Wiley & Sons*, chap.5, 1998.

[13] R. Zimmermann, "VHDL Library of Arithmetic Units", *Forum on Design Languages*, Lausanne, September 1998.

[14] "XC4000E and XC4000X series Field Programmable Gate Array", *Xilinx Inc.*, June 1997.

[15] "Foundation Series 2.1i User Guide", *Xilinx Inc.*, 1999.

[16] "ES2 ECPD07 Library Databook", *Atmel ES2*, September 1997.

[17] "TMS320C3X User's Guide", *Texas Instruments*, June 1992.