An FPGA 2D-convolution unit based on the CAPH language

Abiel Aguilar-González, Miguel Arias-Estrada, Madaín Pérez-Patricio, J L Camas-Anzueto

Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Abiel Aguilar-González · Miguel Arias-Estrada · Madaín Pérez-Patricio · J. L. Camas-Anzueto

An FPGA 2D-convolution unit based on the CAPH language

Preprint version, the final publication and supplementary material are available at http://link.springer.com/article/10.1007/s11554-015-0535-1

Received: date / Revised: date

Abstract Convolution is an important operation in image processing applications, such as edge detection, sharpening, adding blurring and so on. Convolving video streams in real-time is a challenging task for PC systems, however, FPGA devices can successfully be used in these tasks. In this article, the design and implementation of a reconfigurable FPGA architecture for 2D-convolution filtering is described. The filtered frames are calculated at a rate of 103 frames per second for images up to 1200×720 pixel resolution. By using a shift-based arithmetic and circular buffers, the developed FPGA architecture allows to reduce the hardware resources consumption up to 98% compared to the conventional convolution implementations, provides high speed processing and enables to manage large number of different convolution kernels. On the other hand, by using the CAPH language it is possible to reduce the design time up to 75% compared to the plain VHDL design. Furthermore, to maintain high flexibility in concordance with the input video, the developed hardware allows to configure the resolution of the input images with values of $3 \times Y$ up to $1200 \times Y$, and allows scalability for different sizes of convolution kernels of simple and systematic form. Finally, the developed FPGA architecture for the proposed method was implemented and validated in an FPGA Cyclone II EP2C35F672C6 embedded in an Altera development board DE2.

Abiel Aguilar-González (✉) · Miguel Arias-Estrada
Instituto Nacional de Astrofísica, Óptica y Electrónica,
Tonanzintla, Puebla, México.
Reconfigurable computing laboratory.
Tel.: +123-45-678910
E-mail: abiel@inaoep.mx

Madaín Pérez-Patricio · J. L. Camas-Anzueto
Instituto Tecnológico de Tuxtla Gutiérrez,
Tuxtla Gutiérrez, México.
Postgraduate and research department.

1 Introduction

Digital image processing involves procedures that are normally expressed in the form of algorithms. With the exception of image acquisition and representation, most functions of image processing can be implemented in software. The reason for using specialized hardware in image processing is due to need for higher speed processing in several applications such as, low-light microscopy, object recognition, computer vision, robotics and so on [16, 2, 13, 9, 12, 34, 1, 4].

The bus architecture on most computers, except in a few high performance computers with embedded specialized image processing technology (for example CUDA technology or another GPU technology), do not allows to use the data-rate required for real-time applications. Due to high speed requirements most the current real-time image processing systems consist of a combination of computers, peripherals and specialized image processing hardware, such as FPGA devices or GPGPU devices. In most cases, the overall operation is controlled by software running on the host computer or embedded processor when the FPGA or GPGPU has one.[19, 18].

1.1 Spatial-domain convolution filters

2D-convolution is an important concept in several areas of math and engineering including computer vision, digital image processing for medical applications, statistics applications, object recognition and so on [3, 7, 8, 10, 15, 17, 24, 29, 14, 11, 12, 34, 1, 4]. Convolution operates on two signals in 1D filtering operation or two images in 2D filtering operation. In case of 2D-convolution, one image is defined as the input signal or input image and the other, called kernel, is defined as a filter on the input image, producing an output image.

Due to 2D-convolution is the type of convolution used for digital image processing, we will concentrate on the discrete 2D-convolution. 2D-convolution between an image **I** and a convolution mask **M** of order n is defined as shown in the **equation** 1, where $w = (n-1)/2$ [19, 18].

$$C(x,y) = \sum_{u=-w,v=-w}^{u=w,v=w} \mathbf{I}(x+u, y+v) \cdot \mathbf{M}(x+u+w+1, y+v+w+1) \quad (1)$$

By applying the **equation** 1, the convolution operation calculates the intensity value of an image pixel. The resulting pixel is computed considering the pixels from the vicinity and based on the weights of the elements of a spatial filter or kernel whose dimensions are equal to the vicinity of the image to be processed. Each of the elements of the kernel is multiplied with the corresponding value of the vicinity of the original image. Then, the sum of products correspond to the value of the pixel in the new image. Each of the pixels of the vicinity contributes, with its own value and a percentage to calculate the new pixel, **Fig.** 1 [19, 18].

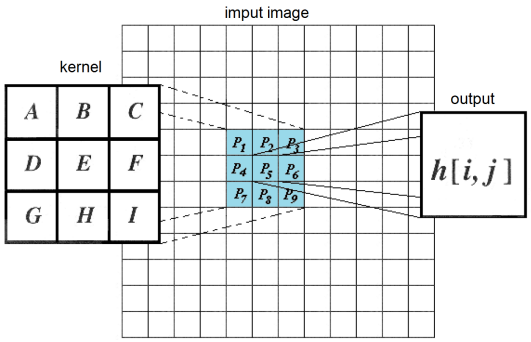


Fig. 1: Convolution process for a 3×3 convolution kernel.

1.2 The CAPH language

CAPH [25, 26, 27, 28] is a domain specific language suited to the description and implementation of stream-processing applications on FPGAs. CAPH relies upon the **actor**/dataflow model of computation. Applications are described as networks of purely dataflow actors exchanging tokens through unidirectional channels. The behavior of each actor is defined as a set of transition **rules** using pattern matching. The CAPH suite of tools currently comprises a reference interpreter and a compiler producing both Systemic and synthesizable VHDL code. These tools allow to reduce the design time and enable high level programing compared with the conventional VHDL design.

1.3 Related works

In aeronautical and automotive applications, image processing systems with high speed processing are required. In order to reduce cost, it is necessary to perform several pre-processing algorithms like 2D convolution in dedicated hardware such as FPGA devices. Therefore, several FPGA implementations for 2D-convolution have been reported in the literature. In [22] it is presented an FPGA edge filter based on an image convolution in which color images are convolved with a pair kernels with quaternion coefficients. When the developed filter is applied to a color image, the filter converts areas of smoothly-varying color to shades of grey and generates colors in regions where edges occur in the image. An FPGA filter based on convolution with hypercomplex masks was introduced by authors of [23]. It was presented three color edge filters inspired by the Prewitt, Sobel and Kirsch filters. The presented filters produce an almost grayscale image with color edges when the original image present a sharp change of color.

Autors of [5] present a two dimensional system approach exploiting dynamic and partial reconfiguration in order to adapt the system architecture to the current requirements of image processing applications. The developed FPGA architecture allows adapting the processing elements as well as the communication infrastructure. The authors performed the convolution operation for a 384×286 pixel resolution images with a 1.69 ms runtime and a 5×5 convolution kernel. Finally, in [6] it is described the design of 2D convolution filters with large kernels, up to 50×50 coefficients, using the Impulse CoDeveloper^{TM} high-level synthesis (HLS) tool. The authors provide a practical guide for designers willing to make the most of an HLS tool like Impulse CoDeveloper, and compare the results, in terms of area utilization, minimum clock period and power consumption, with implementations developed using lower-level design tools. The results show that RTL based implementations can achieve higher performance than CoDeveloper-based ones. However, CoDeveloper can meet the high performance requirements of the most demanding real-time applications, but with less effort and shorter design cycles.

1.4 Motivation

In practice, the convolution operation **eq.** 1 consists of a sum of products. However, when the algorithms are implemented in dedicated hardware such as FPGA devices, a high hardware resource consumption is required. In addition, due to elements of the convolution kernel $\in \mathbb{R}$ the computation of products with fixed point numbers is required, which increases the consumption of the hardware resources.

Even if the IEEE's optimized fixed point libraries are used, the product operation maintains a high hardware resource demand. Hence, it is possible to affirm that the problem resides in the product operation. The high hardware resource demand was our main motivation to search ways in which the product-operation is substituted by other operation which allows simple HDL implementation and enables to process the majority of the kernels using in industrial applications that involves convolution operations.

The rest of this paper is organized as follows: the **section** 2 presents the proposed algorithm for the real-time convolution operation. In the **section** 3, it is detailed the design via the CAPH language for the proposed algorithm's FPGA architecture. In addition, a review and analysis about the CAPH language are shown. Experimental results for different filters, a comparison regarding to different multiplication algorithms reported in the literature and a validation via MatLab is reported in the **section** 4. Finally, the **section** 5 concludes this article.

2 The proposed mhetod

Main objective is to demonstrate three contributions by investigating an FPGA architecture for a 2D-convolution filter described with the CAPH language. First, it is proposed a shift-based arithmetic which allows high flexibility regarding to the convolution operation and enables both reduction of the hardware resources consumption (up to 98%) and high speed processing, appropriate for real-time image processing systems. The filtered images are calculated at a rate of 103 frames per second for images up to 1200×720 pixel resolution. The novelty lies in the design of the FPGA architecture for the proposed method. By the use of strategies such as, the use of shifts-based operators, and kernels coefficients defined as $\pm\frac{\nu_1}{\nu_2}$; it is possible to manage large number of different convolution kernels and maintain low hardware resources consumption. Furthermore, to maintain high flexibility regarding to the input video, the developed hardware allows to configure the resolution of the input images with values of $3 \times Y$ up to $1200 \times Y \quad \forall\, Y \quad \in \quad \mathbb{N}$, allows to configure the kernel order with α values $\forall\, \alpha \quad \in \quad \mathbb{N}$ and allows scalability for different sizes of convolution kernel and input image resolution using a simple and systematic form. Finally, the proposed method performance enables the resulting hardware can be implemented to a wide range of real-time image processing applications.

Second, we study the advantages and disadvantages regarding to the CAPH language and its flexibility with respect to synthesizable VHDL source codes. Third, we provide a practical guide for developers willing to design and implement real-time image processing systems via the CAPH language.

2.1 The shift-based arithmetic

Considering a convolution kernel $\mathbf{M}$ of $(n*2)+1 \times (n*2)+1 \;\forall\; n \quad \in \quad \mathbb{N}$, defined by coefficients in the form $m_{i,j} = \pm\frac{\nu_1}{\nu_2} \;\forall\; \nu \quad \in \quad 2^k$ for $k \in \mathbb{N}, \max(\mathrm{k}) =$ `bits per pixel(bpp)`; all convolution kernel can be expressed as three matrices, $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$, as shown in **equation** 2.

$$m_{i,j} = \sum_{i=1,j=1}^{i=n,j=n} \frac{2^k}{2^k} = \mathbf{A}, \mathbf{B}, \mathbf{C} \tag{2}$$

In the **equation** 3 one kernel and its $\pm\frac{\nu_1}{\nu_2}$ representation is shown, while, in the **equation** 4 the corresponding $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ matrices for the selected kernel are shown. $\mathbf{A}$ is the sum of the k values of the numerator and denominator for the $m_{i,j}$ elements. $\mathbf{B}$ is the sign of the $m_{i,j}$ elements defined by **equation** 5 and $\mathbf{C}$ is defined as shown in **equation** 6.

$$\begin{bmatrix} -1 & \frac{1}{4} & -1 \\ \frac{1}{4} & 8 & \frac{1}{4} \\ -1 & \frac{1}{4} & -1 \end{bmatrix} \quad = \quad \begin{bmatrix} -\frac{2^0}{2^0} & \frac{2^0}{2^2} & -\frac{2^0}{2^0} \\ \frac{2^0}{2^2} & \frac{2^3}{2^0} & \frac{2^0}{2^2} \\ -\frac{2^0}{2^0} & \frac{2^0}{2^2} & -\frac{2^0}{2^0} \end{bmatrix} \tag{3}$$

original mask $\qquad \pm\frac{\nu_1}{\nu_2}$ representation

$$\begin{bmatrix} -\frac{2^0}{2^0} & \frac{2^0}{2^2} & -\frac{2^0}{2^0} \\ \frac{2^0}{2^2} & \frac{2^3}{2^0} & \frac{2^0}{2^2} \\ -\frac{2^0}{2^0} & \frac{2^0}{2^2} & -\frac{2^0}{2^0} \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 3 & 2 \\ 0 & 2 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4}$$

$\pm\frac{\nu_1}{\nu_2}$ mask $\qquad$ **A** $\qquad$ **B** $\qquad$ **C**

$$b(i,j) = \begin{cases} 1, m_{i,j} < 0 \\ 0, m_{i,j} \geq 0 \end{cases} \tag{5}$$

$$c(i,j) = \begin{cases} 1, m_{i,j} < 1 \\ 0, m_{i,j} \geq 1 \end{cases} \tag{6}$$

On the other hand, if $\mathbf{G}$ is an image with λ bits per pixel(bpp), his pixel values can be expressed by λ bit size registers and it is possible to define the $\mathcal{H}$ and $\mathcal{T}$ operators as shown in the **equations** 7 and 8. Where $\mathtt{sll}_\alpha$ and $\mathtt{srl}_\alpha$ corresponds to a left or right shift over a register and the magnitude of the shift is determined by α and `complement` represents the two's complement, applied over a λ bit size register.

$$\mathcal{H}_{i,j,A}^{r,t,C}\{\mathbf{G}\} = \begin{cases} \mathtt{sll}_{(a_{i,j})}(g_{r,t}), \text{ if } c_{i,j} = 0 \\ \mathtt{srl}_{(a_{i,j})}(g_{r,t}), \text{ if } c_{i,j} = 1 \end{cases} \tag{7}$$

$$\mathcal{T}^{r,t}_{i,j,B}\{g_{r,t}\} = \begin{cases} (g_{r,t}), & \text{if } b_{i,j} = 0 \\ \texttt{complement}(g_{r,t}), & \text{if } b_{i,j} = 1 \end{cases} \quad (8)$$

By applying the $\mathcal{H}$ and $\mathcal{T}$ operators, the convolution operation between a $\mathbf{G}$ image and a $\mathbf{M}$ convolution kernel can be expressed by the $\mathcal{C}$ operator as shown in **equation** 9. Where $g_{i,j}$ represents the gray scale value of the (i,j) pixel of the $\mathbf{G}$ image, expressed by one λ bit register. n represents the order of the convolution kernel. $b_{x,y}$ are the elements of the $\mathbf{B}$ matrix for the $\mathbf{M}$ kernel. A, B and C are the $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ matrices for the $\mathbf{M}$ kernel, respectively. & applies a concatenation between registers, and x, y are defined by the **equations** 10 and 11. Finally, the convolution output image $\mathbf{Q}$ for a $\mathbf{G}$ input image with horizontal resolution equal to r_1, vertical resolution equal to r_2 and a $\mathbf{M}$ convolution kernel of type $\pm\frac{\nu_1}{\nu_2}$ can be computed as shown in the **equation** 12.

$$\mathcal{C}\{\mathbf{G}\} = \sum_{u=-n,v=-n}^{u=n,v=n} b_{x,y} \;\&\; \mathcal{T}^{u,v}_{x,y,B}\{\mathcal{H}^{u,v,C}_{x,y,A}\{g_{i+u,j+v}\}\} \quad (9)$$

$$x = \left(\frac{n-1}{2} + 1\right) + u \quad (10)$$

$$y = \left(\frac{n-1}{2} + 1\right) + v \quad (11)$$

$$q(i,j) = \sum_{i=1,j=1}^{i=r_1,j=r_2} \mathcal{C}\{\mathbf{G}\} \quad (12)$$

2.2 Performance and limitations

Before to applying the proposed method it is necessary to define some restrictions and limitations. First, the proposed method only is applicable if all the coefficients of the kernel can be expressed in the form $\pm\frac{\nu_1}{\nu_2}$. i.e., a kernel that includes the 0.173 coefficient cannot be expressed in the form $\pm\frac{\nu_1}{\nu_2}$, while a kernel with coefficients such as, $0.25, 0.50$ or 0.125 are permissible. The use of kernel's coefficients defined by powers of two limits the scope of the proposed method. However, several applications such as gauss filtering, edge filtering, sharpened operation, pyramidal reduction and so on can be successfully applied by using the proposed method. On the other hand, some applications such as Gabor filtering or another directional filter cannot take advantages by applying the proposed method. Second, all the coefficients of the kernel have to be simplified, i.e., an coefficient equal to $\frac{2}{4}$ must be treated as $\frac{1}{2}$, $\frac{4}{12}$ as $\frac{1}{3}$, $\frac{2}{16}$ as $\frac{1}{8}$ and so on. Finally, a consideration for the zero value must be addressed. In all cases the zero value for any coefficient from the kernel will be defined as $\frac{2^0}{2^{\text{bpp}}}$, where (bpp) is the bits per pixel in the input image.

On the other hand, the most important difference and advantage of the proposed method is to avoid the calculation of products and floating point operations, allowing to use large number of different convolution kernels with positive or negative coefficients for integers and fractional numbers. As a result, the proposed method can maintain a high flexibility regarding to the possible different input convolution kernels, near to 1×10^{19} for kernels of order 3. In addition, the proposed method enables parallel implementation, suitable for FPGA devices.

3 FPGA architecture

An overview of the developed FPGA architecture are shown in **Fig.** 2. This architecture consist into three inputs. The values corresponding to the horizontal resolution of the video sequence, size of the convolution kernel and the coefficients of the kernel are sent via logical vectors to the FPGA architecture, `settings` input. The `clk_pixel` input is defined as the pixel rate of the input image while the `input_pixel [7:0]` input is defined as grayscale values of pixels from the input image. On the other hand, the filtered pixels are placed in a logical vector of 8 bits size, `output_pixel [7:0]` output.

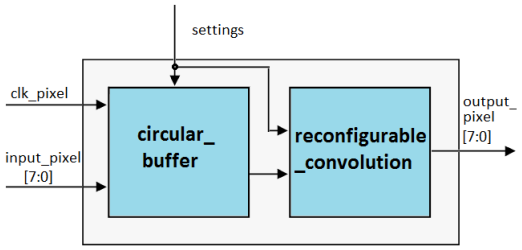


Fig. 2: Black box diagram for the proposed FPGA convolution unit

3.1 The CAPH design

The main difference between conventional HDL design and the CAPH design is the form by define behavior of the modules. In any HDL language such as VHDL or Verilog languages, behavior of the modules is defined via digital approach. Digital approach forces the user knowledge about digital electronics issues such as clock signals, control signals, state machines, RTL description and so on. In addition, in some cases time diagrams and propagation analysis must be applied. Requirements for electronic issues coupled with low level programing presented in the HDL design limits the scope. In most cases electronic, mechatronic or another similar specialists perform the HDL design process.

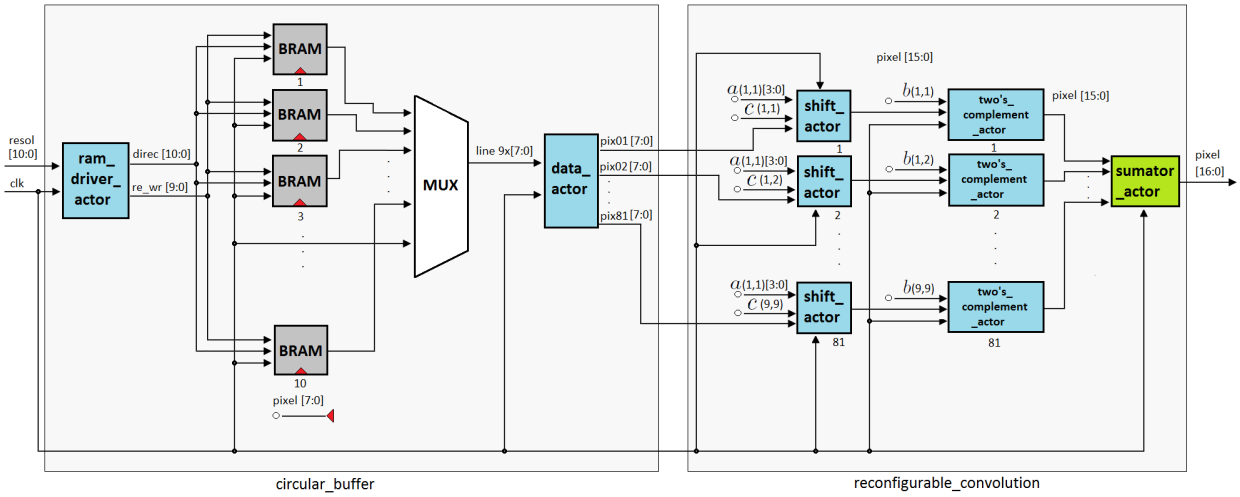


Fig. 3: FPGA architecture for the developed convolution unit

On the other hand, CAPH uses a design based on high level design. High level design avoids digital electronics issues. The modules called actors are defined via transition rules. Each rule defines an HDL process and his control signals. However, signals placement and process definition are performed in automatic form by the CAPH environment. User only requires knowledge about structured programming. In addition, the CAPH programing syntax and operators are similar to the C language, therefore, any user with general programming knowledge could develop FPGA architectures, which increases the scope regarding to the conventional HDL design. Finally, considering the high level design which avoid the electronics optimization stage such as time diagrams and propagation analysis it is possible to decrease the design time.

In the **Fig.** 3 the general diagram for the developed FPGA architecture is shown. The developed CAPH design consist into two stages. In the first stage, the **circular_buffer** stage, stores the necessary data to computing the convolution operation for the input settings. The `resol [15:0]` input is defined as the horizontal pixel resolution for the input frame while the `pixel [7:0]` input contain the grayscale value for one pixel from the input frame and the `clk` input defines the pixel rate. On the other hand, in the second stage, the **reconfigurable_convolution** stage, the pixels and coefficients for the convolution kernel, `pix`$\cdots$ `[7:0]` and `a(i,j)[3:0]`, `b(i,j)` and `c(i,j)`, respectively are used to compute the convolution operation via the proposed method. Finally, the convolution result is placed in the `output pixel` `81`$\times$`[16:0]` output. In the following subsections the details about the developed design are presented.

3.1.1 The circular_buffer stage

All 2D-convolution operation involves passing a 2D kernel over an image, and carrying out a calculation at each position of the kernel. In this way image memory must be accessed several times in order to complete a computation. For processing purposes, the conventional approach is to store the entire input image into a frame buffer, access the neighborhoods pixels and apply the convolution operation needed to produce the output image. If real-time processing is required, considering an n$\times$n kernel, n$\times$n pixel values are needed to perform the computations each time the kernel is moved and each pixel in the image is read up to n$\times$n times. The memory bandwidth constraints make impossible to obtain all pixels stored in the memory in only one clock cycle, unless any kind of local caching is performed.

Conventional approaches are characterized by their abundant memory directly connected to each processing element. However, we proposed the use of a circular buffer schema, [21, 20]. By using pointers of memory address it is possible to keep track of the elements being processed. Input data from the previous n rows can be stored using the memory buffers till the moment when the kernel is scanned along subsequent rows. To reuse hardware resources when new image data is processed, a shift mechanism between buffer rows is used. Data inside the buffer can be accessed in parallel and each input pixel is fed only once to the FPGA device. Buffer elements synchronize the supply of input pixel values to the processing elements. Furthermore, image buffers allow performing several window operators in parallel and enable the possibility to carry out computations with local data. Instead of sliding the kernel across the image, this implementation feeds the image through the kernel as shown in **Fig.** 4.

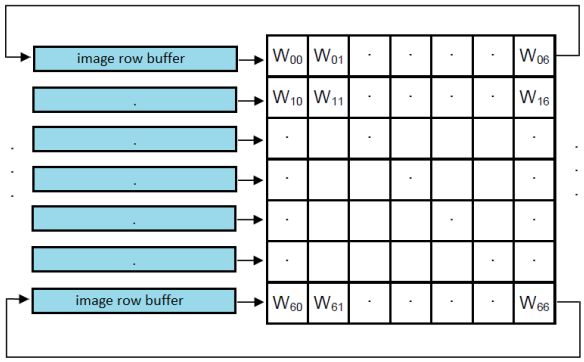


Fig. 4: Structure for the circular buffer

In order to implement the circular buffer scheme in the CAPH language, the use of the `ram_driver` and `data` actors is proposed, **Fig.** 3. Although the circular buffer schema allows to store n horizontal lines and the maximum kernel size could be set as n, design a reconfigurable FPGA architecture with a large maximum kernel size implies extensive source codes. Furthermore, the majority of the industrial applications use kernels sizes with values between 3×3 to 13×13. Hence, for practical purposes, in the developed FPGA architecture the maximum kernel size is set as 9×9, enabling to process kernels up to 9×9.

Considering the maximum kernel size as 9×9, the `ram_driver` actor manages an array of 10 BRAM cores assigning to each one the corresponding address and the write-read value. The BRAM cores consists of a synchronous single-port block-ram unit. This cores were designed via MegaWizard Plug-In Manager - Quartus II. The design settings were assigned as: type = synchronous, width = 8, depth = 1200, operation type = single port and all others parameters were defined as default. These parameters allow to store the gray scale values for each pixel contained in a horizontal line from input images up to 1200 horizontal resolution with 8 bits per pixel (bpp). The use of an BRAM core array enables to read the grayscale values of the pixels contained in n horizontal lines from an image, see **Table** 1.

Table 1: Behavior of the used BRAM array

`w/r [9:0]`	**Read lines by BRAM**$_{1,2,\ldots,10}$
0000000001	-,-,-,-,-,-,-,-,-,-
0000000010	-,-,-,-,-,-,-,-,-,1
0000000100	-,-,-,-,-,-,-,-,2,1
0000001000	-,-,-,-,-,-,-,3,2,1
0000010000	-,-,-,-,-,-,4,3,2,1
0000100000	-,-,-,-,-,5,4,3,2,1
0001000000	-,-,-,-,6,5,4,3,2,1
0010000000	-,-,-,7,6,5,4,3,2,1
0100000000	-,-,8,7,6,5,4,3,2,1
1000000000	-,9,8,7,6,5,4,3,2,1
0000000001	10,9,8,7,6,5,4,3,2,-
0000000010	10,9,8,7,6,5,4,3,-,11

The `ram_driver` actor consist in one input, (`resol`) input, corresponding to the horizontal resolution of the input image and two outputs, `re_wr` and `direc`. The `re_wr` output consist on a logic vector with 10 bits of size, the write-read value of each of the BRAMs are determined by each one of the bits of the logic vector. The `direc` output consist on a logic vector with 11 bits of size that corresponds to the read/write address for all the BRAM cores. The maximum horizontal resolution for the input image is set as 1200 pixels. Hence, the maximum possible value for the `direc` output is 1199.

In the **Fig.** 5 the source code for the `ram_driver` actor is presented. As can be seen, the CAPH programing possesses design sequence like structured programing. The CPAH design sequence could be divided into five stages. In the input/output declaration stage, the input/output definitions for a particular actor must be addressed. In the CAPH environment it is possible to define multiple actors in same source file. Hence, the input/output declaration stage and the second, and third stages must be performed for each actor defined in the file. The second stage addresses the variable declaration used into an actor. Then, in third stage the transition rules for an actor must be defined. After, in the fourth stage the input/output for the general CAPH design must be specified. Finally, in the fifth stage, the instantiation process between actors defined in the source code could be performing.

```
actor ram_driver
in      (
        resol   :       unsigned<11>
        )
out     (                                                          ①
        re_wr   :       unsigned<10>,
        direc   :       unsigned<11>
        )
--------------------------------------------------------------------
var esta :unsigned<11>
var cont :unsigned<11>
var re_w :unsigned<10>                                             ②
--------------------------------------------------------------------
rules
| (resol:x,esta:1,cont:y,re_w:z)-> (direc : if y < x-1 then y+1 else 0,
                                    cont  : if y < x-1 then y+1 else 0,
                                    re_wr : if y < x-1 then z else if z
                                    < 258 then z << 1 else 1,      ③
                                    re_w  : if y < x-1 then z else if z
                                    < 258 then z << 1 else 1)
| (resol:x)                      -> (cont:x,esta:1,re_w:512);
--------------------------------------------------------------------
stream input     : unsigned<11> from "entrada01.txt";
stream salida1   : unsigned<10> to   "salida1.txt";                ④
stream salida2   : unsigned<11> to   "salida2.txt";
--------------------------------------------------------------------
net (salida1,salida2) = ram_driver (input);                        ⑤
```

Fig. 5: Source code for the `ram_driver` actor

The **ram_driver** actor consists into two transition rules. The first, increments the initial value of the read/write address for all the BRAM cores, and manages the `re_wr` and `direc` outputs, **Fig.** 3. The second rule, assigns initial values for the variables used in the **ram_ram_driver** actor. Due to each BRAM core only provides the pixel value of one pixel of the horizontal line stored inside them, in order to access the others horizontal values necessary for the convolution operation, it is proposed the use of the **data** actor, **Fig.** 3. The **data**

actor consists into one transition rule. Applying this, the values of one pixel for the nine lines are read in parallel at any time instant and placed in the outputs 1-9 of the **data** actor, then this values are placed in the outputs 10-18 and the new read data are placed in the outputs 1-9. This process are repeated until storage the 81 pixels for the 9×9 selected kernel size is completed.

3.1.2 The reconfigurable convolution stage

The **reconfigurable_convolution** stage, **Fig.** 3, is comprised of three different actors. First, $n \times n$ **shift** actors performs the corresponding shifts regarding to each of pixels in the kernel, **equation** 7. The values of the **A** and **C** matrices serve as inputs of the **shift** actors, `a(i,j)[3:0]` and `b(i,j)`, respectively. In order to perform the right shift over large grayscale values for the input pixels, all input pixel values are placed in registers of 16 bits of depth. **shift** actor consists into two rules, this rules represent the two possible solutions of the **equation** 7, **sll** or **srl**. Then, $n \times n$ **two's_complement** actors perform the corresponding complements regarding to each pixels in the kernel, **equation** 8. The values of the **B** matrix and the outputs of the **shift** actors serve as inputs of the **two's_complement** actors. The **two's_complement** actor consists into two transition rules, the rules represent the two possible solutions of the **equation** 8. Considering the size of the input pixels as 16 bits, the **equation** 8 can be simplified as shown in **equation** 13. After, all values generated by the **two's_complement** actors are concatenated with the coefficients of the **B** matrix. Finally, the **adder** actor performed the sum of all input pixels via one signed-sum unit defined in one transaction rule.

$$\mathcal{T}^{r,t}_{i,j,B}\{g_{r,t}\} = \begin{cases} (g_{r,t}), & \text{if } b_{i,j} = 0 \\ \texttt{complement}(g_{r,t}), & \text{if } b_{i,j} = 1 \end{cases}$$

$$= \begin{cases} (g_{r,t}), & \text{if } b_{i,j} = 0 \\ 65536 - (g_{r,t}), & \text{if } b_{i,j} = 1 \end{cases} \quad (13)$$

3.2 Instantiation

CAPH environment performance regarding to instantiation is limited. In general, it is possible to instantiate between actors contained in the same source file. However, this process is complex in comparison with the design of the actors and considering the CAPH 2.3 version (released in July 2014) , the version used in this research, it is possible indicate that CAPH not allow instantiation for actors defined in different source files. In addition, when the source code of the actors is considerable, define multiple actors in the same source file is not practical.

Due to limitations regarding to instantiation process, it is proposed the use of a VHDL instantiation file. All actors were compiled via the CAPH compiler and it was obtained the following files: complement_ act.vhd, data_act.vhd, ram_ driver_act.vhd, shift_act.vhd and adder _act.vhd. Furthermore, MegaWizard Plug-In provides the bram.vhd file. All files have been instantiated in a VHDL file designed in Quartus II Web Edition 10.1SP1.

In order to assign lines in the outputs of the `data` actor in ascending form, i.e. `line1 [7:0]` = input image line number l, `line2 [7:0]` = input image line number l+1, $\cdots$, `line9 [7:0]` = input image line number l+8, the outputs from the BRAM cores in read mode are assigned to the outputs of the `data` actor as seen in **Table** 2, the first column corresponds to the output `w/r [n+1:0]` of the `ram_driver` actor, the second column corresponds to the numbers of the BRAM cores assigned to the outputs of the `data` actor while in **Fig.** 3, the instantiations between the BRAM cores and the **data_actor** is shown.

Table 2: Input assignment for the `data` actor

`w/r [n+1:0]`	**Assignment** `lineK [7:0] for K=1,2,` $\cdots$ `,9`
0000000001	2,3,4,5,6,7,8,9,10
0000000010	3,4,5,6,7,8,9,10,1
0000000100	4,5,6,7,8,9,10,1,2
0000001000	5,6,7,8,9,10,1,2,3
0000010000	6,7,8,9,10,1,2,3,4
0000100000	7,8,9,10,1,2,3,4,5
0001000000	8,9,10,1,2,3,4,5,6
0010000000	9,10,1,2,3,4,5,6,7
0100000000	10,1,2,3,4,5,6,7,8
1000000000	1,2,3,4,5,6,7,8,9

4 Results and discussion

4.1 CAPH's performance and limitations

CAPH language possesses several advantages regarding to the conventional HDL design. First, the CAPH language takes advantages regarding to the conventional HDL languages due to the possibility to avoid digital approach which involves time requirements and control signals definitions. In addition CAPH allows a high level programing and enables to export to synthesizable VHDL code the source code. In addition, CAPH allows to use most of the synthesizable VHDL operators such as shifts, arithmetic operations and so on. Other advantage for to the CAPH language is the optimized operators for arithmetic operations. Several arithmetic operators such as `+,-,*,/` and `sqrt`, produced VHDL optimized codes. It is useful when complex arithmetic operations such as products, quotients and radicals need to be implemented in FPGA devices.

CAPH performance regarding to instantiation is limited. In general, it is possible to instantiate between actors contained in the same source file. However, this process is complex in comparison with the design of the actors and considering the CAPH 2.3 version (released in July 2014), it is possible indicate that CAPH not allow instantiation for actors defined in different source files. In addition when the source code of the actors is considerable, define multiple actors in the same source file is not practical. Furthermore, although the majority of the CAPH instructions are exportable to VHDL, other limitation are operations between floating points and disability to low level programing operations such as, concatenation between registers and operations between bits of a register.

4.2 General performance of the developed FPGA architecture

The main characteristics and differences between the developed FPGA architecture and all the FPGA convolution architectures reported in the literature are two. First, the possibility to process different resolution regarding to input video stream without have to re- synthesize the developed hardware. Second, the possibility to applying different sizes or coefficients for the convolution kernel without have to re-synthesize the developed hardware. In the **Fig.** 6 the coefficients-distribution of the used convolution kernel is shown. As can be seen the central pixel corresponds to the 41 index. All the possible kernels must be centered in the central pixel, e.g., a 3×3 kernel must contain the pixels with the indexes equal to 31-33, 40-42 and 49-51, while a 5×5 kernel must contain the pixels with the indexes equal to 21-25, 30-34, 39-43, 48-52, and 57-61.

01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81

Fig. 6: Distribution of the coefficients for the convolution kernel

4.3 Simulation results

All modules and cores for the FPGA architecture in **Fig.** 3 were compiled and synthesized in Quartus II Web Edition version 10.1SP1. In order to verify functionality of all the modules individually, post-synthesis simulations in ModelSim-Altera 6.6c were executed. In order to evaluate the behavior of proposed algorithm, the developed FPGA architecture was simulated using different convolution masks and different input image sizes. For this purpose a VHDL testbench was designed. In this testbench the pixels that integrate one frame of a video sequence was read each rising edge of a 100 MHz clock, the settings corresponding to horizontal resolution and convolution mask were set in test signals, the filtered pixels were stored in a txt file. Finally, the txt output file was decoded by Matlab and the filtered frame was stored in a .jpg image format. The testbench file was ran in ModelSim-Altera 6.6c. The resulting images are shown in **Fig.** 7. In a first test, an unitary kernel was applied, **Fig.** 7b. In a second test, an edge detector filter [23] was implemented, **Fig.** 7e. In a third, a sharpened operation was applied, as shown in **Fig.** 7h. Finally, the fourth test consists in a Gaussian blurring operation [6], **Fig.** 7k. Due to it were tested both multiple input resolutions and different sizes and kernel's coefficients (included rational, positive and negative numbers), **Fig.** 7 demonstrates the flexibility for the developed FPGA architecture regarding to the convolution operation.

In addition, to validate the functionality for each of the four tests presented in the **Fig.** 7, a convolution script that operates considering the **equation** 1 was designed in MatLab. The output images for the same convolution kernels are stored in a .jpj file. Then, the percentage of the number of erroneous pixels (ϕ) and the RMS error (σ) were computed by the **equations** 14 and 15; where I_1 is the output image generated for the conventional convolution method. I_2 is the output image generated for the proposed convolution method. x is the horizontal resolution of the input image. y is the vertical resolution of the input image. N is set as $x * y$ and Δ_ψ is set as 0.01. In the **Fig.** 8 the results of this validation are shown.

$$\phi = \frac{1}{N} * 100 \sum_{i=1,j=1}^{i=x,j=y} |I_1(i,j) - I_2(i,j)| > \Delta_\psi \tag{14}$$

$$\sigma = \sqrt[2]{\frac{1}{N} \sum_{i=1,j=1}^{i=x,j=y} (I_1(i,j) - I_2(i,j))^2} \tag{15}$$

(a) original image 512×512

(b) filtered image

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(c) convolution kernel

(d) original image 200×200

(e) filtered image

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(f) convolution kernel

(g) original image 200×200

(h) filtered image

$$\begin{bmatrix} -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} \\ \frac{1}{4} & 2 & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} \end{bmatrix}$$

(i) convolution kernel

(j) original image 512×512

(k) filtered image

$$\begin{bmatrix} 0 & 0 & 0 & \frac{1}{128} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{128} & \frac{1}{64} & \frac{1}{128} & 0 & 0 \\ 0 & \frac{1}{128} & \frac{1}{64} & \frac{1}{8} & \frac{1}{64} & \frac{1}{128} & 0 \\ \frac{1}{128} & \frac{1}{16} & \frac{1}{8} & \frac{1}{4} & \frac{1}{8} & \frac{1}{16} & \frac{1}{128} \\ 0 & \frac{1}{128} & \frac{1}{64} & \frac{1}{8} & \frac{1}{64} & \frac{1}{128} & 0 \\ 0 & 0 & \frac{1}{128} & \frac{1}{64} & \frac{1}{128} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{128} & 0 & 0 & 0 \end{bmatrix}$$

(l) convolution kernel

Fig. 7: Performance of the developed FPGA convolution unit

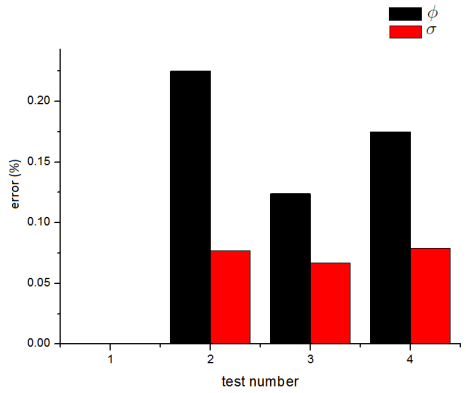


Fig. 8: Validation of the developed FPGA convolution unit

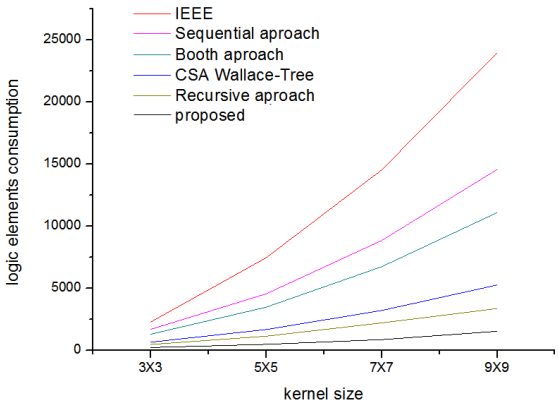


Fig. 9: Comparison of logic elements consumption

By analyzing the **Fig.** 8 it is possible to affirm the correct functionality of the FPGA architecture. The first test, **Fig.** 7 (a)-(c), involves anything operation due to the selected kernel is the identity kernel. In this case both errors σ and ψ must be zero as shown in **Fig.** 8, therefore it is possible to affirm the correct functionality of the developed FPGA architecture. On the other hand, due to decimal representation of the tests performed in MatLab in contrast to the integer representation used in the ModelSim-Altera simulations, small error values for the second-fourth tests (**Fig.** 7 (d)-(l)) is expected. However, due the σ and ψ values obtained for these tests, close to zero, it is possible to ensure the correct functionality of all actors and cores in the FPGA architecture.

In order to evaluate the performance of the developed FPGA architecture, several comparisons between different approaches for the conventional convolution method are presented. For this purpose the developed FPGA architecture (**Fig.** 3) was modified as shown in **Fig.** 10. The modification consist in replace any **shift_actor** and **two's_ complement_actor** by one **multiplication_ actor**. The **multiplication_ actor** performs the multiplication operation between the pixels and coefficients in the convolution kernel, `pix···[7:0]` and `k (x,y)`, respectively. Due to conventional multiplication stage and the proposed reconfigurable convolution stage do not require memory bits consumption, only comparisons between logic elements requirements were conducted, **Fig.** 9. IEEE approach consists into multiplication modules designed using the specialized fixed point libraries developed by IEEE. Sequential multiplication modules presented by Stevenson [31] are implemented in Sequential approach. Booth approach uses the multiplication module presented by Takagi et al [33]. In CSA Wallace-Tree approach, the schema proposed by S.Wallace [32] is implemented. Finally, in Recursive approach, it is used the architecture proposed by Singh-Parihar and Reddy [30] which consist in an optimized low level multiplication unit.

By analyzing **Fig.** 9 it is possible to affirm that even if the most efficient multiplication algorithms are used, the multiplication operation maintains a high hardware resources consumption. However Due to the proposed method consists into a shift-based arithmetic any FPGA device can take advantages by applying shift operations. Shift operations are low level operations which involves only one digital element in contrast with multiplication operations which involves several digital elements such as adders and logical elements (and or and so on), which increase the hardware resources consumption. As can be seen, if the proposed method is applied the hardware resources requirements can be reduced up to 98%. Reduction regarding to the hardware resources requirements is very useful in autonomous applications such as robotic applications, where the use of small FPGA devices that implies relatively few hardware resources is needed. On the other hand, any real-time application will take advantages if the hardware resources usage decreases.

Due to, in most of the image processing systems such as, low-light microscopy, object recognition, computer vision, and so on [3, 7, 8, 10, 15]; a high speed processing is required, other parameter considered to evaluate performance is speed processing. In **Table** 4, processing speed comparisons between the same approaches presented in the **Fig.** 9 and the proposed method are presented. On the other hand, in **Table** 3 processing speed comparisons for the developed architecture regarding to different input image resolution are shown. As can be observed, the proposed **reconfigurable_convolution** stage allows to maintain processing speed close or equal to the processing speed of all the compared multiplication methods. Furthermore, the developed FPGA architecture enables high speed processing for different resolutions of input video streams, **Table** 3. Hence, it is possible to affirm that the developed architecture is appropriate for real-time image processing applications. Finally, in **Table** 5 the full hardware resource consumption for the developed FPGA convolution unit is shown.

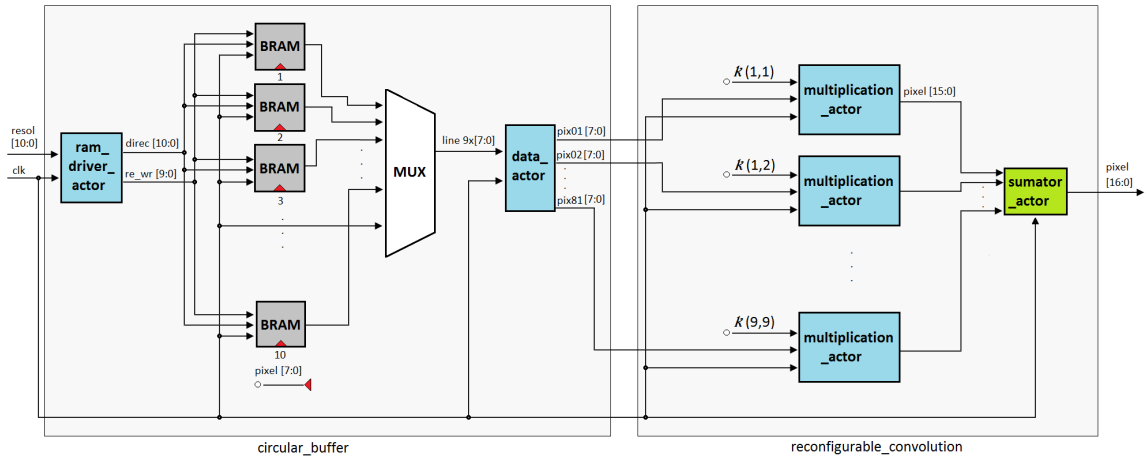


Fig. 10: FPGA architecture for the conventional convolution method

Table 3: Processing speed for different resolutions of input images

Method	Resolution	Frames/s	Pixels/s
IEEE	1200×720	103	135,005,160
S.Wallace [32]	1200×720	103	135,005,160
Singh-Parihar and Reddy [30]	1200×720	104	136,614,880
Stevenson [31]	1200×720	104	136,614,880
Takagi et al [33]	1200×720	104	136,614,880
proposed*	1200×720	103	135,005,160
proposed*	512×512	526	137,887,744
proposed*	200×200	3500	140,000,000

*Operating frequency = 100 MHz

Table 4: Processing speed for different convolution methods

Method	Required time	Clock specification
IEEE	10.0 ns	100 MHz
S.Wallace [32]	10.0 ns	100 MHz
Singh-Parihar and Reddy [30]	9.61 ns	100 MHz
Stevenson [31]	9.54 ns	100 MHz
Takagi et al [33]	9.54 ns	100 MHz
proposed	10.0 ns	100 MHz

Table 5: Hardware resource consumption for the developed FPGA convolution unit

Resource	Demand
Total logic elements	8,573
Total combinational functions	8,573
Dedicated logic registers	3,351
Total pins	523
Total Memory Bits	163,840
Embedded multiplier elements	0
Total PLLs	0

4.4 Implementation results

4.4.1 Post-synthesis implementation

The developed architecture was implemented in an FPGA Cyclone IIEP2C35F672C6 embedded in an Altera development board DE2. The maximum clock frequency was defined as 50 MHz. In order to validate the implementation, the **frame_generator** and the **settings** modules were designed, **Fig.** 11. The **frame_generator** module have stored in this one frame of 200×200 pixels resolution and provides grayscale values from the stored frame, `pixel [7:0]`. The **settings** module provides the horizontal resolution value for the input frame, `settings [15:0]` and the coefficients for the convolution kernel, `kernel 81x[5:0]`. The architecture that includes the **frame_generator** and the **settings** modules was im-

plemented in the IIEP2C35F672C6 FPGA device and was simulated via post-synthesis simulation performed in ModelSim-Altera 6.6c. The output image is stored in a text file. Then, the ψ and σ errors (**equations** 14 and 15) are computed in MatLab, considering the output image for the FPGA implementation and the output image for the same settings performed in MatLab. The obtained error values for the ψ and σ variables were 0 and 0, respectively. This values confirms the correct functionality of the implementation. In **Fig.** 12 the specifications for the FPGA implementation are shown.

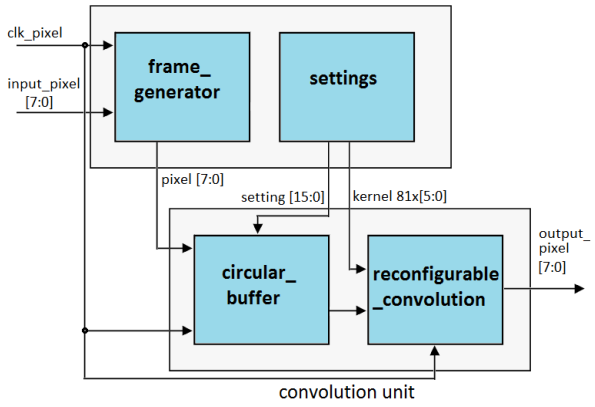


Fig. 11: FPGA architecture for the post-synthesis implementation

(a) original image 200×200 (b) filtered image

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(c) convolution kernel

Fig. 12: Specifications for the post-synthesis implementation

4.4.2 Real-time implementation

In order to ensure real-time processing for the developed FPGA architecture, other FPGA implementation with maximum clock frequency set as 50 MHz. was performed, **Fig.** 13. A TRDB DC2 board was connected in the first port of expansion of the DE2 as video acquisition device. TRDB DC2 board provides images of 1280×1024 pixel resolution in RGB scale at rate of 30 frames per second. To determine the value in gray scale of the input images the value of the green channel is used as gray scale value. The filtered output for the selected kernel was shown in a 4,3" LCD screen of 800×480 pixel resolution of the terasIC brand connected to the second port of expansion of the DE2 board, **Fig.** 14. In this implementation first, the **acquisition** module provides grayscale values from pixels contained in frames from an input video stream, `pixel [7:0]` output. Then, the developed **convolution** unit performs the convolution operation for the input settings. The input settings (horizontal resolution and convolution coefficients, `settings [15:0]` and `kernel 81x[5:0]`, respectively), were defined by the switchs and buttons from the DE2 board. Finally, the filtered pixels, `output_pixel [16:0]`, were sent to the LCD display.

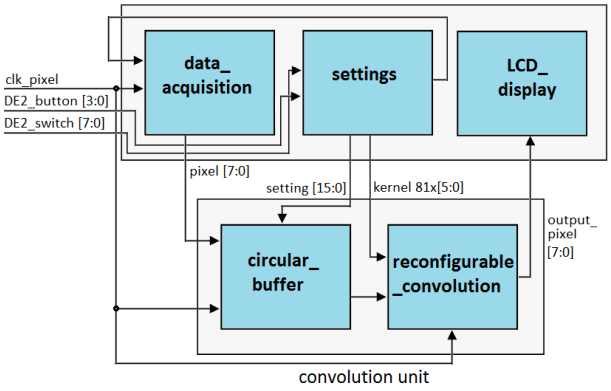


Fig. 13: FPGA architecture for the Real-time implementation

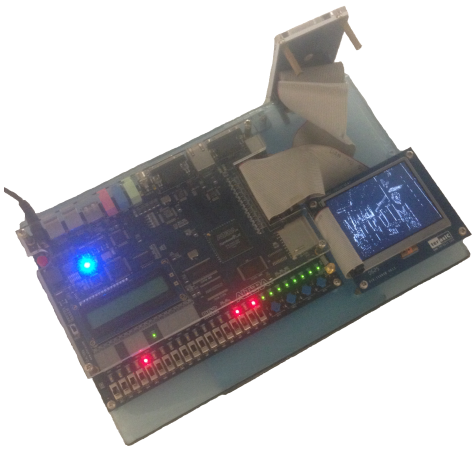


Fig. 14: Real-time implementation

5 Conclusions

In this article a new method for the 2D-convolution operation in real-time has been presented. In this method, floating point operations are translated into low level operations between registers. Low level operations allow to reduce the hardware resources consumption, near to 98% compared to the conventional convolution implementations. In addition, the developed FPGA architecture was designed via the CAPH compiler. Due to high level programing and avoid to use electronic digital approach that involves time diagrams and propagation analysis, CAPH enables to reduce design time near to 75% compared to the plain VHDL design. The developed FPGA convolution unit allows to maintaining high flexibility regarding to the possible different input convolution kernels, near to 1×10^{19} for kernels of order 3, 7×10^{179} for kernels of order 9 and high speed processing. High speed processing and high flexibility regarding to the convolution kernel enable the developed convolution unit to be implemented to a wide range of real-time image processing applications. To validate functionality, post-synthesis simulations of the developed FPGA architecture has been performed. Furthermore, the architecture was implemented in an FPGA Cyclone II EP2C35F672C6 embedded in a development board DE2 of Altera. In all cases high accuracy level have been observed.

One purpose of this research was to explore the potential of the CAPH language for complex image processing tasks, in concordance, the FPGA architecture for the proposed method was performed in the CAPH environment which offers great advantages regarding to synthesizable VHDL design. Only one important limitation was detected in this research, flexibility for instantiation process. However, a VHDL instantiation file designed in the Altera VHDL tools or another VHDL design tool, easy solves this inconvenience.

Finally, an important characteristic of the presented architecture is scalability. All the actors, sub-actors and cores that integrate the FPGA architecture easily allow adaptation to work with convolution kernel of size major than the 9×9 kernel, and horizontal video resolution larger than the 1200 resolution which were the values used in the developed FPGA architecture. Consequently, it is possible to configure a wide variety of filters with appropriate values to the characteristics of the input video sequence and the user requirements.

References

1. Acharya KA, Babu RV, Vadhiyar SS (2014) A real-time implementation of sift using gpu. J Real-Time Image Proc
2. Asgher U, Muhammad H, Hamza H, Ahmad R, Butt S, Jamil M (2013) A temporal superresolution method applied to low-light cardiac fluorescence microscopy. In: Proceedings of The 2013 Asilomar Conference on Signals, Systems and Computers, IEEE, Pacific Grove, CA, pp 1073–1077
3. Asgher U, Muhammad H, Hamza H, Ahmad R, Butt S, Jamil M (2014) Robust hybrid normalized convolution and forward error correction in image reconstruction. In: Proceedings of The 10th International Conference on Innovations in Information Technology, IEEE, Al Ain, United Arab Emirates, pp 54–59
4. Barina D, Zemcik P (2015) Vectorization and parallelization of 2-d wavelet lifting. J Real-Time Image Proc
5. Braun L, Gohringer D, Perschke T, Schatz V, Hubner M, Becker J (2009) Adaptive real-time image processing exploiting two dimensional reconfigurable architecture. J Real-Time Image Proc 4:109 125
6. Colodro-Conde C, Toledo-Moreo F, Toledo-Moreo R, Martínez-Álvarez J, Garrigós-Guerrero J, Ferrández-Vicente J (2014) A practical evaluation of the performance of the impulse codeveloper hls tool for implementing large-kernel 2-d filters. J Real-Time Image Proc 9:263279
7. Fiack L, Cuperlier N, Miramond B (2013) Embedded and real-time architecture for bio-inspired vision-based robot navigation. J Real-Time Image Proc
8. Fons F, Fons M, Cantó E, López M (2009) Real-time embedded systems powered by fpga dynamic partial self-reconfiguration: a case study oriented to biometric recognition applications. J Real-Time Image Proc 8:229251
9. Hofmann M, Eggeling C, Hell SJS (2005) Breaking the diffraction barrier in fluorescence microscopy at low light intensities by using reversibly photoswitchable proteins. Proceedings of the National Academy of Sciences of the United States of America 42:17,565–17,569
10. Jiang B, Woodell A, Jobson D (2014) Novel multi-scale retinex with color restoration on graphics processing unit. J Real-Time Image Proc
11. Krause M, Alles RM, Burgeth B, Weickert J (2013) Fast retinal vessel analysis. J Real-Time Image Proc
12. Krause M, Alles RM, Burgeth B, Weickert J (2013) Fast star centroid extraction algorithm with sub-pixel accuracy based on fpga. J Real-Time Image Proc
13. M Arias Estrada CTH (2000) Real-time fpga arquitectures for computer vision. In: Proceedings of The Electronic Imaging 2000-Photonics West, dedicated conference on Machine Vision Applications in Industrial Inspection VII, San Jose, CA., pp 23–28
14. Mabrouk A, Hassim N, Elshafiey I (2013) A computationally efficient technique for real-time detection of particular-slope edges. J Real-Time Image Proc
15. Park H, Park Y, Oh SK (2003) L/m-fold image resizing in block-dct domain using symmetric convolution. IEEE Transactions on Image Processing 12:1016–1034
16. Rasnik I, French T, Jacobson K, Berland K (2013) Electronic cameras for low, light microscopy. ELSEVIER ACADEMIC PRESS INC, San Diego
17. Reichenbach SE, Geng F (2001) Improved cubic convolution for two dimensional image reconstruction. IEEE Nuclear Science Symposium and Medical Imaging Conference 3:1775–1778
18. Romero-Troncoso R (2004) Sistemas digitales con VHDL. Legaria, México
19. Romero-Troncoso R (2007) Electrnica Digital y Lógica Programable. Universidad De Guanajuato, México
20. Saldaa G, Arias-Estrada M (2006) Customizable fpga-based architecture for video applications in real time. In: Proceedings of The IEEE International Conference on Field Programmable Technology, IEEE, Bangkok, Thailand, pp 381–384
21. Saldaa G, Arias-Estrada M (2007) Compact fpga-based systolic array architecture suitable for vision systems. In: Proceedings of The 4th International Conference on Information Technology: New Generations, IEEE, Las Vegas, Nevada, pp 1008–1013
22. Sangwine S (2002) Colour image edge detector based on quaternion convolution. Electronics Letters 10:969–971
23. Sangwine S, Ell T (2002) Colour image filters based on hypercomplex convolution. IEE Proceedings - Vision, Image and Signal Processing 147:89–93
24. Savarimuthu TR, Kjaer-Nielsen A, Sorensen AS (2011) Real-time medical video processing, enabled by hardware accelerated correlations. J Real-Time Image Proc 6:187197
25. SEROT J (2012) Caph : A high-level actor-based language for programming fpgas. In: Workshop on Architecture of Smart Cameras - WASC 2012
26. SEROT J (2013) Caph: A domain specic language for implementing stream-processing applications on recongurable hard. In: First Workshop on Domain Specific Languages Design and Implementation, URL http://dsldi2013.hyperdsls.org/
27. SEROT J, BERRY F (2013) Caph, un langage dé dié á la synthése; applications flot de données sur circuits fpga. In: 24eme Congrés

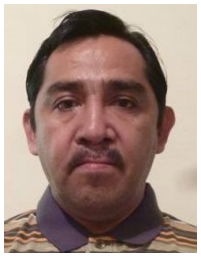
GRETSI
28. SEROT J, BERRY F, AHMED S (2012) CAPH: A Language for Implementing Stream-Processing Applications on FPGAs, vol Embedded Systems Design with F, Springer, chap CAPH: A La, pp 201–224. URL http://link.springer.com/chapter/10.1007/978-1-4614-1362-2_9
29. Shi J, Reichenbach S (2006) Image interpolation by two-dimensional parametric cubic convolution. IEEE Transactions on Image Processing 54:1857–1870
30. Singh-Parihar RK, Reddy S (2005) Efficient Floating Point 32-bit single Precision Multipliers Design using VHDL. BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI 333031, Pilani
31. Stevenson D (1981) A proposed standard for binary floating point arithmetic. IEEE Transactions on Electronic Computers 14:51–62
32. SWallace C (1984) A suggestion for fast multipliers. IEEE Transactions on Electronic Computers 13:1417
33. Takagi N, Yasuura H, Yajima S (2006) High-speed vlsi multiplication algorithm with a redundant binary addition tree. IEEE Transactions on Electronic Computers 34:789–796
34. Zhou F, Zhao J, Ye T, Chen L (2014) Accelerating embedded image processing for real time: a case study. J Real-Time Image Proc

Author Biographies

Abiel Aguilar-González received the B.Eng. degree of Mechatronics in June 2012, Universidad Politécnica de Chiapas, Tuxtla Gutiérrez, México. In June 2015 he received the M.Sc. degree of mechatronics engineering with highest honors, Instituto Tecnológico de Tuxtla Gutiérrez, Tuxtla Gutiérrez, México. He is currently pursuing his Ph.D. degree in Computer Science at the reconfigurable computing laboratory of the Instituto Nacional de Astrofísica Óptica y Electrónica, Cholula, México. His research interests are mainly real-time image processing, real-time FPGA-based system design, machine learning and fuzzy logic applications.

Miguel Arias-Estrada obtained his B.Eng. in Communications and Electronics, and his M.Eng in Digital Systems at the FIMEE (University of Guanajuato) in Salamanca, Gto. in 1990 and 1992 respectively. In 1998, he obtained his Ph.D. degree at the Computer Vision and Systems Laboratory of Universit Laval (Quebec city, Canada). He was a professor-researcher at the Computer and Systems Laboratory at Laval University where he worked on the development of a Smart Vision Camera. Since 1998 he is with the Computer Science department of INAOE (National Institute of Astrophysics, Optics and Electronics, Puebla, Mexico) where he continues his research on FPGA architectures for computer vision. His interests are Computer Vision, FPGA and GPU algorithm acceleration for 3D and machine vision.

Madaín Pérez-Patricio received the Ph.D. degree of Automation and industrial computing 2005, Université Lille 1 : Sciences et Technologies, France. Since september 1997 he is research professor in department of postgraduate and research, Instituto Tecnológico de Tuxtla Gutirrez, México. His primary research interest include computer vision and reconfigurable computing.



Jorge Luis Camas Anzueto received the PhD degree from Instituto Nacional de Astrofísica Óptica y Elctrónica (INAOE), Puebla, México in 2004. He is currently a researcher in Maestría en Ciencias en Ingeniería Mecatrónica (MCIM) of the Instituto Tecnológico de Tuxtla Gutiérez, Chiapas, México. His research interests include optical sensors, fiber sensors, and optoelectronics.