Journal of Systems Architecture 58 (2012) 277–285

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

FPGA-based architecture for the real-time computation of 2-D convolution with large kernel size

F. Javier Toledo-Moreo *, J. Javier Martínez-Alvarez, Javier Garrigós-Guerrero, J. Manuel Ferrández-Vicente

Dpto. Electrónica y Tecnología de Computadoras, Universidad Politécnica de Cartagena, Spain

ARTICLE INFO

Article history:
Received 15 November 2011
Received in revised form 28 April 2012
Accepted 14 June 2012
Available online 26 June 2012

Keywords:
2-D Convolution
Large kernel size
FPGA
Embedded and real-time systems

ABSTRACT

Bidimensional convolution is a low-level processing algorithm of interest in many areas, but its high computational cost constrains the size of the kernels, especially in real-time embedded systems. This paper presents a hardware architecture for the FPGA-based implementation of 2-D convolution with medium–large kernels. It is a multiplierless solution based on Distributed Arithmetic implemented using general purpose resources in FPGAs. Our proposal is modular and coefficient independent, so it remains fully flexible and customizable for any application. The architecture design includes a control unit to manage efficiently the operations at the borders of the input array. Results in terms of occupied resources and timing are reported for different configurations. We compare these results with other approaches in the state of the art to validate our approach.

1. Introduction

Bidimensional convolution is a basic tool in many areas. Convolution with a kernel or template has been widely used in image or video processing for ages for spatial domain filtering in low-level processing stages with the aim of manipulating data, extracting information of interest, enhancing or nulling specific characteristics, for patterns recognition or other higher-level goals [1,2]. Nowadays more than ever, the ubiquity of cameras in any context in devices like smartphones and tablets, surveillance systems, automobiles, etc. has given rise to a plethora of embedded video processing systems for nearly every conceivable application, and 2-D convolution is a fundamental operation in most of them. Moreover, 2-D convolution is also on the basis of machine learning algorithms [3] or biologically-inspired models [4–7].

For some applications like simple edge detection, image smoothing or sharpening, convolution kernels from small 3×3 up to 7×7 are typically fair enough, as evidences the 5×5 customizable mask for image filtering available in the so-popular user-level Adobe Photoshop® software. For other applications like object tracking, estimation filtering, physiological modeling or pattern recognition, larger kernels can be more interesting (e.g. [8–12]).

Although conceptually simple, the computation of 2-D convolution, given by the sum of products (1), is not trivial: with a $M \times N$ kernel, it requires $M \times N$ multiplications and $M \times N - 1$ additions, besides $M \times N$ accesses to the input data, for the calculation of a single output.

$$O(x,y) = \sum_{i=-\frac{M}{2}+1}^{\frac{M}{2}} \sum_{j=-\frac{N}{2}+1}^{\frac{N}{2}} h(i,j)I(x+j,y+i) \quad (1)$$

This implies that more than 1.35 Giga-operations per second (GOPs) are required to support a real-time processing rate of 1280×720 30 frames per second (fps) HD video with a 5×5 kernel, more than 12.41 GOPs with a 15×15 one. Clearly, the computational load and the complexity of memory access grow exponentially as the kernel dimensions increase.

Historically, the kernel size has been constrained to a small bounded range because of the inefficiency of CPU-based software approaches to carry out such a huge number of operations. Transformation into the frequency domain and computing the convolution as the inverse FFT of the multiplication of the FFTs of both input and kernel is a significant improvement for software implementation, but only through parallel processing it is possible to achieve the desired throughput for real time applications. FPGA devices, and more recently GPUs, offer the chance.

In last few years, Field Programmable Gate Array (FPGA) devices have been the predominant hardware platform used to compute 2-D convolution due to their fine grain parallelism and reconfigurability. FPGA internal structure makes itself perfectly suitable for successfully exploiting pixel-level parallelism inherent to low-level image processing algorithms, like the local neighborhood function defined by Eq. (1), instruction-level parallelism by means of pipelining, as well as, at higher level, task parallelism (e.g. multiple

* Corresponding author.
E-mail address: javier.toledo@upct.es (F. Javier Toledo-Moreo).

1383-7621/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2012.06.002

convolutions in parallel). In fact, low-level image and video processing have been significant application drivers for reconfigurable computing since the early 90s [13].

This context, with 2-D convolution receiving great interest from application designers while computational challenges must be faced, explains why this is still a topical subject, as a number of recent papers reveals. These papers present works proposing novel architectures for optimizing area/performance trade-off [14,15] or for offering higher design/operation flexibility [16–20]. Some of them pursue to reduce the complexity of computations or data buffering by analyzing the operations and data involved [21–23], the separability of filters [24] or properties of their coefficients [25]. Some others implement transformations to recode coefficient representation [24,26] or deal with the implementation, rather than a single filter, of a set of 2-D filters [4,27].

This paper extends the work presented in [28] and proposes an architecture for the FPGA implementation of 2-D convolution. Based on Distributed Arithmetic [29,30], this architecture makes it feasible the convolution with medium–large size kernels at high-resolution video rate. It is coefficient–kernel independent, so neither restrictions or particular properties are required for the coefficients, nor the design do take any advantage of the constancy of coefficients to optimize operations or datapaths, so it remains fully flexible. As another novel contribution of this work, specific circuitry has been included to properly manage the computations at the input borders, an important issue specially when kernel size is large with respect to the input array size.

Results are reported for kernel sizes up to 50×50. To best of the authors' knowledge, there is no work in the literature describing a real-time implementation of such a size 2-D convolver. The bibliographic search leads to a 22×22 kernel with quadrant symmetric coefficients as the largest one, at the same time that it reveals that although most papers define general $M \times N$ kernels, only implement simple 3×3 finally. None of the works mention a special treatment at the borders.

The reported results are for both 1- and 8-bit data. Convoluting a 1-bit binary image with a template is of interest, mostly but not only, in feature extraction and object tracking, using shaping matching. For more general application domains, 8-bit is the standard resolution for raw image data. A range of coefficient values between 2- and 8-bit has been considered for the different configurations. The architecture does not impose any restriction to the number of bits for data or coefficients, or to the values represented by such numbers of bits.

Further the presented configurations, the modularity of our proposal facilitates the customization for any kernel size and for any data or coefficient resolutions. Rather than an ad-hoc solution, our proposal defines a core for the computation of 2-D convolution.

FPGA has been chosen as the hardware platform because of its suitability for embedding processing system. The enhanced characteristics of modern FPGA devices, featured with millions of gates of programmable logic and with dedicated hardware resources, allows to build complete systems with improved performance and reduced costs. Moreover, because of its flexibility, it is possible to implement not only specific algorithms but also interfaces, controllers, even microprocessors, and then to integrate the whole embedded system in just one system-on-a-chip, more powerful and complex.

The architecture is detailed and featured as developed in the following sections. Since 2-D convolution is mathematically the sum of multiple 1-D convolutions, first the hardware design proposed for the computation of $1 \times M$ convolution with 1-bit input data is described in Section 2. Next, it is extended to bidimensional $M \times N$ case, again with 1-bit input data, in Section 3. Design alternatives for n-bit input data are presented in Section 4. Section 5 focuses on the logic control aimed to avoid the output distortions which large kernels can give rise to. Resources and timing results for different configurations are given in Section 6, as well as comparison with other approaches and platforms. Finally, Section 7 concludes the paper.

2. Basic architecture for $1 \times N$ convolution and 1-bit input

Most 2-D convolution implementations rely on the multiplying units embedded in modern FPGAs to carry out all the multiplications in parallel and to achieve great performance. However, as the kernel size increases, the number of embedded multipliers needed grows exponentially. This fact can constrain the kernel size or force to use a bigger FPGA device, which, in its turn, can yield a very high cost per operation ratio. On the other hand, a lot of work has been done on the design of multiplierless filters, mostly in the one-dimensional domain, and some authors have implemented 2-D convolution by replacing multiplications with shifting and adding operations [26] or transforming the computation into the logarithmic domain [25].

Our proposal has been developed over the fundamental concept underlying fine-grain FPGAs operation: an n-input Look Up Table (LUT) can implement any combinatorial function of n variables of 1 bit. With 1-bit resolution input data, the mathematical expression (2) that computes the convolution of the input x with a N-coefficient filter $\{h_0, h_1, \ldots, h_{N-1}\}$ can output 2^N different results.

$$y_n = \sum_{l=0}^{N-1} h_l x_{n-l} \tag{2}$$

If these values are stored in a 2^N-word memory, where the address m stores the convolution result for the input $\{m_0, m_1, \ldots, m_{N-1}\}$ with $m = \sum_{b=0}^{N-1} m_b 2^b$, the result of the convolution for any combination of input values is present at the memory data output port when those $\{m_0, m_1, \ldots, m_{N-1}\}$ values are used to address the memory. This scheme does not need any multiplying unit and performs the computation just with the delay due to memory access time.

Although theoretically extendable to any number of coefficients, the exponential growth of the memory size with N evidences that this basic approach becomes inefficient or even unfeasible for the implementation of relatively small filters in off-the-shelf FPGA devices: 20-tap filter requires a 1-M memory, 1 G for 30-tap filter.

However, this constraint can be easily overcome through the decomposition of Eq. (2) indicated in (3):

$$y_n = \sum_{j=0}^{\lceil \frac{N}{k} \rceil} \sum_{i=0}^{k-1} h'_{kj+i} x_{n-(kj+i)} \tag{3}$$

with

$$\begin{aligned} h'_{kj+i} &= h_{kj+i} \quad \text{if } kj+i < N \\ h'_{kj+i} &= 0 \qquad \text{otherwise} \end{aligned} \tag{4}$$

Under this configuration of table partitioning [31], with the N data clustered into k-data groups, the 2^N-depth memory is no longer necessary but n_{ROM} memories of p_{ROM}-depth each:

$$n_{\mathrm{ROM}} = \left\lceil \frac{N}{k} \right\rceil \tag{5}$$

$$p_{\mathrm{ROM}} = 2^k \tag{6}$$

Each memory is customized for a subset of coefficients, being the content at address m of the memory j determined by:

$$d_{\mathrm{ROM}_j}(m) = \sum_{i=0}^{k-1} h'_{kj+i} m_i \tag{7}$$

with $\{m_0, m_1, \ldots, m_{k-1}\}$ the bits, from the least to the most significant, coding the value of $m, m \in [0, p_{\text{ROM}} - 1]$ and $j \in [0, n_{\text{ROM}} - 1]$.

Regarding data memory width, each memory does store the sum of products of k coefficients with k 1-bit data, so the b_{ROM}-bit word size of each memory must satisfy:

$$b_{\text{ROM}} = \lceil log_2(k \max\{c_i\}) \rceil \text{ if } c_i \in [0, \mathbb{N}) \tag{8}$$

in order to ensure that any possible resulting value can be represented. For $c_i \in \mathbb{R}$, the binary point and the sign must be properly managed.

This table partitioning scheme requires an adder tree to calculate the overall result from the partial sub-convolution results. It has been designed with n_{SUM} adders distributed along $\lceil log_2(n_{\text{ROM}}) \rceil$ levels, with $n_{\text{SUM}}(i)$ adders at the i-level.

$$n_{\text{SUM}} = n_{\text{ROM}} - 1 \tag{9}$$

$$n_{\text{SUM}}(i) = \left\lfloor \frac{\left\lceil \frac{n_{\text{ROM}}}{2^i} \right\rceil}{2} \right\rfloor \tag{10}$$

By default, the adders are fully pipelined to maximize instruction-level parallelism and therefore they achieve maximum speed. Datapath optimizations that may compromise the flexibility of the architecture to manage any resolution for data and coefficients have not been considered in the design of the adder tree.

According to Eq. (2), N data are required to compute the convolution: the input data at each instant and the $N - 1$ previous data. In the architecture, these data are available in a *window buffer* made of $N - 1$ registers in cascade. This window buffer serves as fully customized cache where data items are stored only the time instants they are needed. For a 20-tap filter configuration, Fig. 1 shows the 19 data registers $D_{[1,19]}$ in cascade, with d_{in} the input to D_1 and the D_i input connected to the D_{i-1} output. They all share the same enable and global synchronous reset signals. The size of these registers is fixed by the incoming data resolution: with 1-bit input it minimizes to just a flip-flop.

The D_i outputs, grouped into k-data clusters, address the n_{ROM} ROM memories. For the first memory ROM1, the bus address is made of the value at the input d_{in} and of the outputs of the $k - 1$ first flip-flops, from least to most significant bit respectively. For the remaining memories, the ROMi address is made of the outputs of the flip-flops $D_{k(i-1)}$ to D_{ki-1}, again from least to most significant. With this configuration all the data items needed for the computation are read out in the same cycle, all the ROMs are addressed at the same time and, therefore, all the operations are performed in parallel in only one clock cycle.

The window buffer, the n_{ROM} memories and the adder tree make up the basic architecture of the core for the computation of convolution. The scheme is depicted in Fig. 1 for a 20-tap filter and 4-data clusters configuration, with the additional logic explained in Section 5 included.

3. Basic architecture for $M \times N$ convolution and 1-bit input

The convolution of a $P \times Q$ input array with an $M \times N$ kernel given by (1) is just an extension to the bidimensional space of (2). Consequently, the convolution with an $M \times N$ kernel can be computed with M modules $1 \times N$, each one customized for the suitable coefficients, and an adder tree to obtain the final result from the partial result of each i-row. The adder tree is made of $m_{\text{SUM}} = M - 1$ two-operand adders distributed in $\lceil log_2(M) \rceil$ stages in cascade, and with $m_{\text{SUM}}(i)$ adders at the i-stage:

$$m_{\text{SUM}}(i) = \left\lfloor \frac{\left\lceil \frac{M}{2^i} \right\rceil}{2} \right\rfloor \tag{11}$$

Fig. 2 shows the scheme for a $5 \times N$ configuration, being each i-row module an instance of the component in Fig. 1. Again, the adder tree is fully pipelined.

This scheme allows the window buffers to address all the memories simultaneously, extending to its maximum the parallelism at iteration level and performing all the operations in only one clock cycle. In each clock cycle, the convolution is centered on a new input data, the one available at the $D_{\frac{N}{2}}$ of the row $f_{\frac{M}{2}}$. This is at expense of the latency, given by (12), which is determined by the number of cycles needed for the input data to reach the $D_{\frac{N}{2}}$ flip-flop of the row $f_{\frac{M}{2}}$, plus the latency associated to the pipelining in the adder trees.

$$N\left(\frac{M}{2} - 1\right) + \left(\frac{N}{2} - 1\right) + \left\lceil log_2 \frac{N}{k} \right\rceil + \lceil log_2(M) \rceil \tag{12}$$

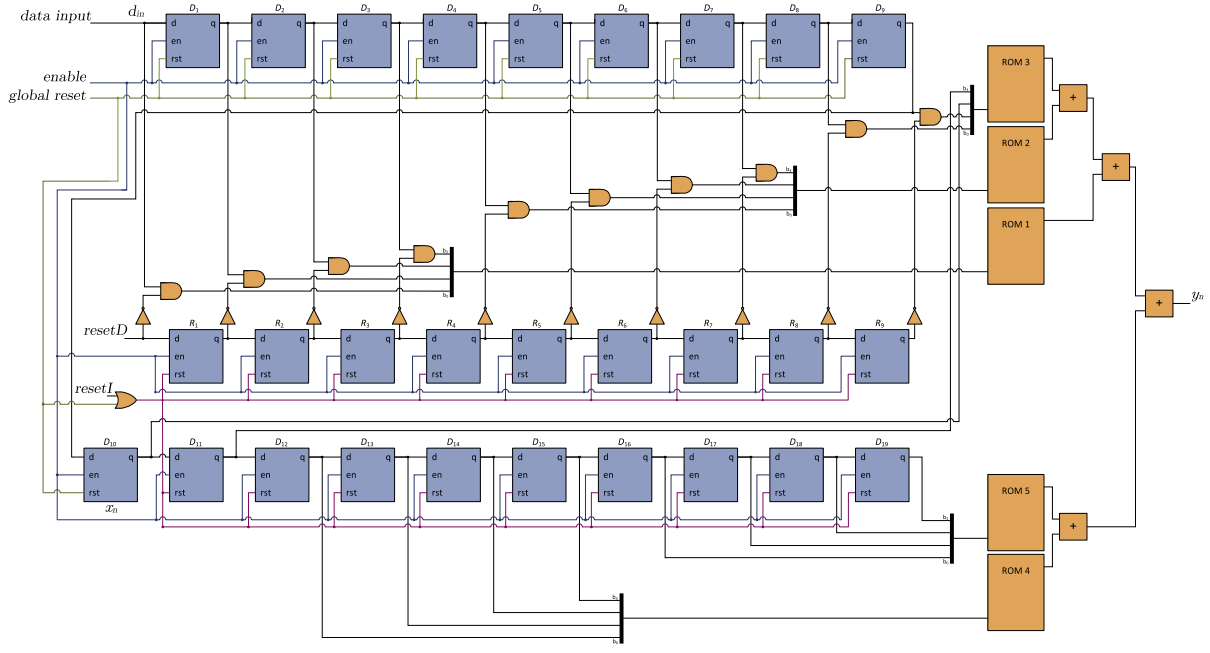

Fig. 1. Scheme of the proposed architecture for 1×20 kernel and 1-bit data input.

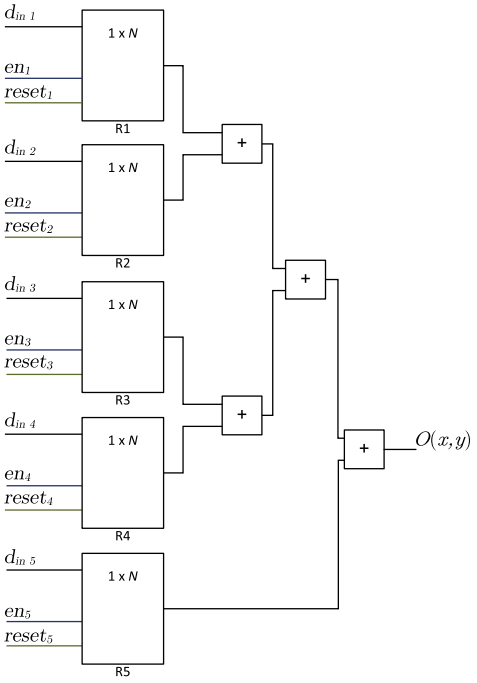

Fig. 2. Scheme for $5 \times N$.

4. Basic architecture for $M \times N$ convolution and n-bit input

The 1-bit design can be easily extended to n-bit input data through a range of approaches, from fully parallel to fully serial [31]. The parallel scheme is shown in Fig. 3. For each row, it consists of n identical instances of the module in Fig. 1 whose outputs are all added according to the weights of their corresponding bit in the input word. Clearly, this solution will provide the highest speed at the expense of the highest resources consumption.

The fully serial scheme of a row is depicted in Fig. 4. Computations are carried out serially at bit-level in the input word using n-bit shift registers that replace flip-flops in the window buffer. The shift and add unit accumulates the partial results according to their corresponding weights in the input data item. With this configuration the increase of the resources occupancy is kept to a minimum, but the convolver throughput is reduced by a factor n, since n clock cycles are required to generate each output result.

Intermediate solutions with a trade-off between speed and resources can be implemented. The final configuration will depend on the requirements of the application and the constraints of the hardware platform.

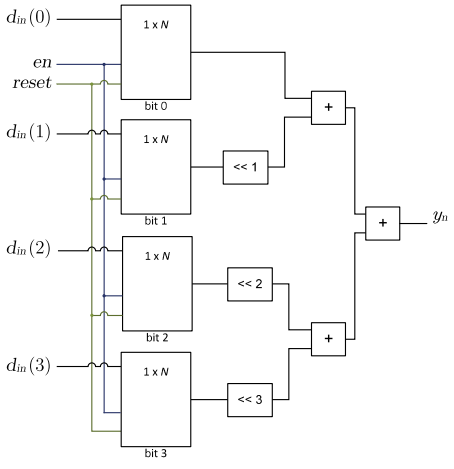

Fig. 3. Parallel scheme for $1 \times N$ kernel and 4-bit data input.

5. Control of operations in the input borders

Eq. (1) yields situations where the kernel does not overlap the input array completely. When convolution is centered on $x \in [1, N/2-1]$ or $x \in [Q-N/2+1, Q]$, positions beyond the borders of the array are pointed, and, as they do not actually exist, ad-hoc values must be assigned to them. As stated in [1], the usual solution in software programming is to consider an extended $(P+M-1) \times (Q+N-1)$ input array padded with zeros.

Meanwhile, hardware solutions typically do not manage this border problem. Although data streaming implies that operations in the array borders are eventually performed on data items that do not correspond to the right indexes in the summatories, the small size of kernels usually implemented reduces its impact on the final result to the negligible. Thus, to avoid the need for special treatment of the border data, only the results inside the so-called Region of Interest (ROI), a sub-array $N/2$ data smaller than the input array in horizontal direction and $M/2$ in vertical direction, are considered as valid [32]. However, with medium-large kernels, specially when the ratio of image and kernel sizes is low, this ROI may result too smaller and the convolution may be useless. This is what happens, for example, in [33], where cropping implies a ROI around a 25% narrower and a 31% lower than the original input image. Therefore, it does become necessary to manage properly the operations in the border of the input. As up-resizing the input array through zero padding is far from being efficient in hardware design, the basic architecture described has been completed with logic aiming a double purpose: on the one hand, to make null the value addressing the ROM memories from positions of the window buffer associated at each instant to any index out of the input array; on the other hand, to preserve the data stored in those positions that must be made null since they are going to be necessary in successive computations of the convolution.

This additional logic controls the values of the data preceding and succeeding the convolution central data item. It consists of a cascade of R_i flip-flops fed with the *resetD* and *resetI* signals and whose outputs make null or not the D_i output values at the memories address buses by using a set of logic gates and with the functionality described next. In operation, when the convolution is centered on an item $I(x,y)$ far enough from the borders, *resetD* and *resetI* remain deactivated (low logic level) and hence the values of the central data, the $N/2-1$ preceding data and the $N/2$ succeeding data, available respectively in $D_{\frac{N}{2}}, D_{[\frac{N}{2}+1,N-1]}$ and d_{in}, and $D_{[1,\frac{N}{2}-1]}$, determine the memories addresses where the sub-convolution partial results are read from. When computation is centered on $(P-N/2+1, y)$, the *resetD* signal is activated (high logic level) and so the value in d_{in}, which actually is $I(1, y+1)$ due to the sequential data streaming, is made null at the corresponding bit of the memory address port. In the following computations, each step more D_i store data after the center of convolution that do not belong to the convolution operations and their outputs must be made null, until the extreme situation when convolution is centered on $I(Q,y)$ and all the $D_{[1,\frac{N}{2}-1]}$ output values must be null at the corresponding memories ports. During this time, *resetD* is kept activated to make null d_{in} while its value goes along the R_i cascade making null the associated D_i outputs. At the following instant, the next computation is centered on $I(1, y+1)$ and the context changes drastically. Now, all the data income after the central one are required for the operations and therefore the $D_{[1,\frac{N}{2}-1]}$ output values must be restored. This functionality is achieved by deactivating *resetD* and resetting the R_i by means of the activation of *resetI*. At the same time, all the data preceding the central one

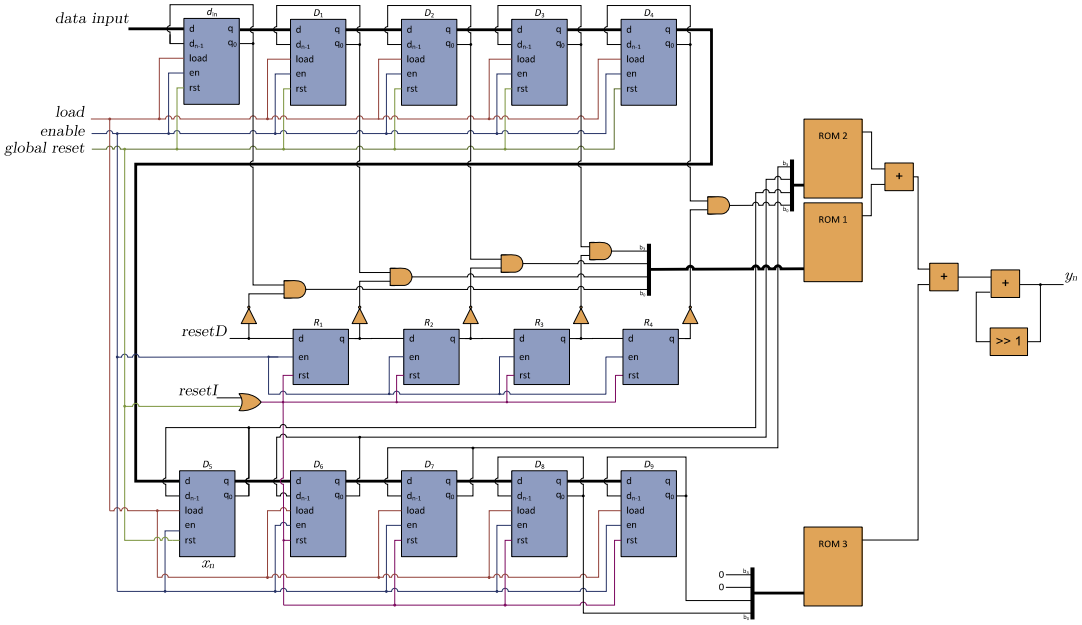

Fig. 4. Serial scheme for 1×10 kernel and n-bit data input.

are not of interest for the operations and must be discarded. This is also managed by *resetI*, which reset the $D_{[\frac{N}{2}+1,N-1]}$ registers of the window buffer. This simple solution implements the zero padding technique with minimum hardware cost and without extra clock cycles.

In general, considering N as even, the border logic control requires $N/2-1$ *flip-flops*. The input to R_1 is *resetD* and the input to R_{i+1} is the output of R_i. All the flip-flops share *resetI* as synchronous reset, which is also connected to the synchronous reset ports of the $D_{[\frac{N}{2}+1,N-1]}$ flip-flops. Signal *resetD* remains activated for those convolutions centered in $x \in [Q-N/2+1, Q]$ and deactivated otherwise. Signal *resetI* is activated only when the convolution computation is centered on $x = Q$, deactivated at low level otherwise. In addition, $N/2$ AND logic gates and as many NOT gates are needed. The outputs of the ANDs are connected to the memories ports associated to those data that follow the central one in the input streaming. The inputs of the first AND are connected to d_{in} and, through a NOT, to *resetD*; the outputs of the remaining ANDs are connected to $D_{[1,\frac{N}{2}-1]}$ and, again through the corresponding NOT, to $R_{[1,\frac{N}{2}-1]}$.

6. Implementation results

The resources occupied in the implementation are determined by the kernel size $M \times N$, the data and coefficients resolutions and the data packing factor k. Among these variables, the first three ones are on the application domain, while the data packing factor can be managed by the hardware designer. Expressions (5), (6), (8) and (9) reveal that increasing k keeping N constant leads to fewer memories, and consequently fewer branches and levels in the adder tree. In contrast, it also entails successive duplications of the memory depth and, eventually, a wider memory word in order to keep the range of values of the coefficients. Since the memories are implemented in a distributed manner using general purpose logic resources, an estimation of the number of n-input LUTs is given by:

$$M \cdot 2^{k-n}\, b_{\mathrm{ROM}} \cdot n_{\mathrm{ROM}} \tag{13}$$

the one associated to the border control logic:

$$M\left(\frac{N}{2}+1\right) \tag{14}$$

and the one consumed by the adder tree:

$$M \sum_{i=1}^{\lceil log_2 n_{\mathrm{ROM}} \rceil} n_{\mathrm{SUM}}(i) \cdot (b_{\mathrm{ROM}} + i - 1) + \sum_{j=1}^{\lceil log_2 M \rceil} m_{\mathrm{SUM}}(j) \cdot (b_{\mathrm{ROM}} + \lceil log_2 n_{\mathrm{ROM}} \rceil + j - 1) \tag{15}$$

On the other hand, the flip-flops making up the window buffer, the flip-flops in the borders control logic and the pipelining stages in the adder trees sum up the overall number of flip-flops consumed:

$$M\left(\frac{3}{2}N-2\right) + M \sum_{i=1}^{\lceil log_2 n_{\mathrm{ROM}} \rceil} \left\lceil \frac{n_{\mathrm{ROM}}}{2^i} \right\rceil (b_{\mathrm{ROM}} + i - 1) + \sum_{j=1}^{\lceil log_2 M \rceil} \left\lceil \frac{M}{2^j} \right\rceil (b_{\mathrm{ROM}} + \lceil log_2 n_{\mathrm{ROM}} \rceil + j - 1) \tag{16}$$

Fig. 5 plots graphically these estimations of resources for 4-input LUT (left) and 6-input LUT (right) FPGAs, corresponding to Xilinx Virtex-4 and previous families, and to Virtex-5 and newer ones, respectively. Regarding LUTs, it is clear that a minimum occupancy is reached if the data cluster size k matches with the number of LUT inputs characterizing the device. This fact is due to the preponderance of the LUT-based ROM memories implementing the sub-convolutions. Regarding flip-flops, their number decrease as k increases due to the reduction of the adder tree size. However, it is noteworthy that with k being n-input the number of flip-flops come closer to the number of LUTs than in any other configuration, which can simplify the routing process and lead to better timing results due to the 1:1 rate of both types of general purpose resources in FPGA slices design. Thus, it is considered the best configuration to adjust k to the number of inputs of the FPGA LUTs.

The impact of k on b_{ROM} in order to support full output resolution given a coefficient resolution, defined in (8), is shown in Fig. 6.

These expressions allow to estimate the maximum resources required for a configuration and to evaluate the impact of each parameter, but the occupancy results ultimately rely on the optimizations that can be carried out given a particular set of coeffi-

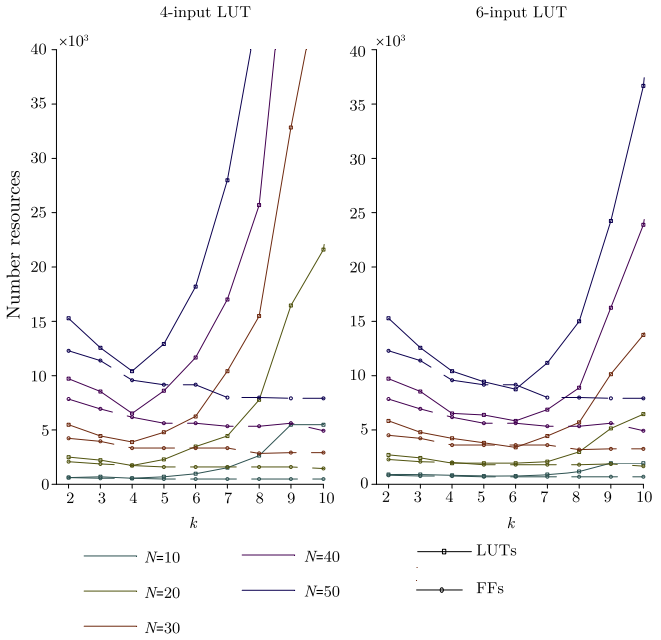

Fig. 5. Estimation of LUTs and flip-flops needs versus k, for 4-bit $N \times N$ kernels in 4-input and 6-input FPGAs.

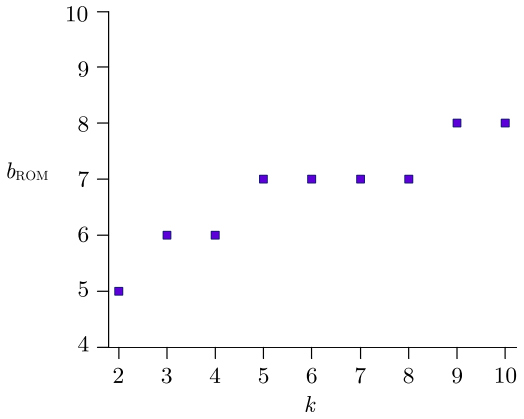

Fig. 6. Evolution of b_{ROM} with k for 4-bit coefficients.

cients. In accordance with those values, both the designer can customize the datapath, and the synthesis and *Place*&*Route* tools can trim unnecessary logic.

The proposed architecture has been designed using Xilinx System Generator, and synthesized and implemented on Virtex-4 with Xilinx ISE10.1. System Generator has been of great use to the validation of the 2-D convolution results.

Table 1 summarizes the average values of occupied resources and the minimum clock periods of different configurations with random kernels, 1-bit input and b_{ROM} with values 4 and 10, which implies 2- and 8-bit coefficients respectively.

Post-place&*Route* timing results show the high throughput of the proposed solution. It is possible, for instance, to perform the convolutions described in [33] of a binary image with a 30×30 2-bit kernel in 4.65 ns per input pixel, which yields more than 215 millions of output data per second. For the largest 50×50 size, it only takes 5.01 ns to calculate one data, which means nearly 1 tera-operations per second, assuming continuous data streaming and no limitation in the input data bandwidth. These data reveal that parallelism and pipelining reduces to a minimum the influence of increasing M or N on T_{min} at the expense of a higher resources occupation. The clock

Table 1
Occupied resources and timing for different configurations implemented on Virtex-4 FPGA with 1-bit data.

b_{ROM}	4			10		
Kernel size	20×20	30×30	50×50	20×20	30×30	50×50
Flip-flops	1101	1599	6911	1809	2577	11159
LUTs 4-input	1262	1810	7581	2560	3704	15675
Area (*slices*)	1156	1712	7154	1801	2673	11221
T_{min} (ns)	4.37	4.65	5.01	4.47	4.68	5.43

cycle cost is, in fact, an increase of latency according to (12). A similar behavior is confirmed when evaluating different coefficient resolutions through the change of parameter b_{ROM}, as examplified with a 30×30 kernel and 8-bit data configuration shown in Fig. 7.

Given b_{ROM} bits, these implementation results are independent of the format of the numbers to be represented: integer or fixed-point. Xilinx System Generator, the tool used to design and evaluate the architecture, allows to define fixed-point data by specifying the total number of bits and the location of the binary point. It is the hardware designer who determines b_{ROM} and the position of the binary point to accommodate the integer part of the possible resulting values and to achieve the acceptable precision for the fractional part. The arithmetical operators instantiated in the model have been configured with full precision, so the output data size is adjusted to represent the result without error.

Table 2 summarizes resources and *Post-place*&*Route* timing values for 8-bit data with both fully parallel and fully serial approaches. Coefficients are 8-bit resolution. The increase of the resources needs hinder the routing process and slows down the speed processing slightly with respect to 1-bit input, but, again, parallelism and pipelining provided by the architecture allow to reach very high throughputs. It is worthy to highlight that the presented results ensure real time 2-D convolution of 1920×1080 video at 60 fps with any of the presented kernel sizes.

6.1. Study of pipelining

The way the architecture has been presented, fully pipelined, maximizes the instruction-level parallelism and leads to maximum processing speed. However, depending on the application requirements, such a high speed could be not so necessary and the number of pipelining stages at the adder trees could be reduced, which results in area savings at the expense of longer combinatorial paths and hence higher minimum period. Table 3 shows the impact of pipelining at the full structure of adder trees, from the ROM memories output data ports to the output. The number of LUTs occupied is not affected and it is not included.

6.2. Comparison with other approaches and platforms

As mentioned in Section 1, works have been done to develop new solutions for accelerating or optimizing the computation of 2-D convolution. Although most of them generalize their design for $M \times N$ kernels, only a few present real results for implementations of kernels larger than 3×3. For these ones, Table 4 gathers together information about the kernel size, resolutions, throughput and FPGA device. For comparison, Table 4 also includes implementation results for configurations of the proposed architecture not reported previously and for designs based on embedded multipliers.

First, it is to highlight that no other work presents such large 2-D convolution implementation like this work. Zhang et al. [25] describe an architecture for 22×22 kernels with quadrant symmetric property and transformation of values into the logarith-

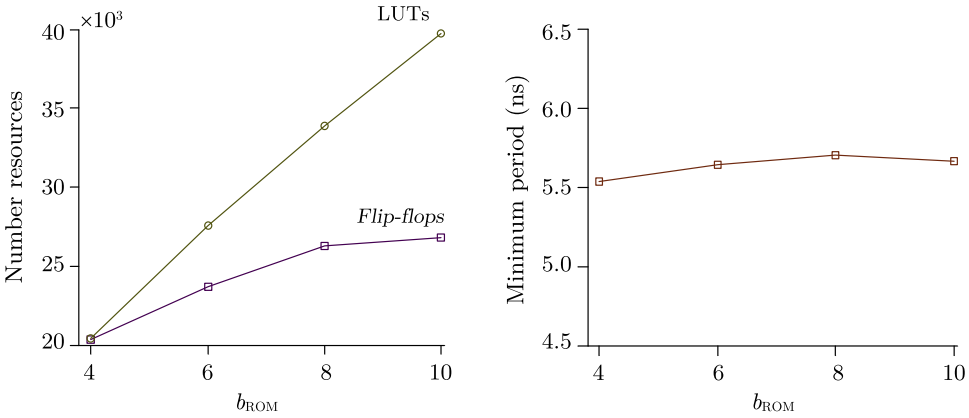

Fig. 7. Influence of b_{ROM} parameter on resources and timing.

mic domain to avoid multiplications; our proposal is nearly two times faster and without any need of special properties on the coefficients values. The 22×22 convolution with embedded multipliers is also slower than our approach. It is the largest kernel size that can be implemented with embedded multipliers in Virtex-4 family. Our proposal also outperforms the medium-size implementations collected in Table 4. Farabet et al. [5] and Wong et al. [36] offer higher throughput for 7×7 and 7×5 respectively, as well as the 7×7 with embedded multipliers, but these small kernel sizes is out of the scope of this work.

It is also noteworthy that none of these works describes a solution for managing the operations at the input array borders and avoiding the output distortions. The designs with embedded multipliers do not either manage the borders problem, which would make more complex the place and route process and would reduce the maximum throughput.

The great emergence of Graphics Processing Units (GPUs) over the last few years has changed the scene of high performance computing and real time processing. In spite of the difficulty to fairly compare FPGAs and GPUs because of the continuous changing in technology and marketing strategies of manufacturers of both platforms, work is being undertaken with this purpose. Particularly, 2-D convolution is considered a useful testbench due to its high degree of parallelism, data dependencies and high demanding memory access requirements. In the recent comparison in [34], it is claimed that parallel operation on Virtex-4 FPGA provides higher speed processing than GeForce7900 GPU from 4×4 kernels and on. According to their results, the herein presented implementation for 11×11 kernel outperforms the GPU by 3×, growing exponentially as kernel size increases. [34] also reveals that the GPU performance degrades sharply with window size, which contrasts with our FPGA approach where the impact of window size is much lower and is mostly assumed by the device occupation. For newer GPUs, [37] concludes that FPGA is better suited than GeForce GTX580 to implement 11×11 Gabor filterbanks. A key advantage for FPGAs is the choice of the degree of parallelism to implement, as well as the full customization of the datapath. Moreover, the smaller clock frequency and power requirements imply that FPGAs offer better GOPS per unit of power ratio than GPUs, an important issue when developing embedded systems. Beyond special case of 2-D convolution, which suits better for FPGA features, [38] has proven that FPGA outperforms high-end Nvidia GPU in applications that do match a GPU in execution model and map well to a GPU architecture.

Table 3
Implementation results for different pipeline depths. Rate 1:*x* means one level of adders with pipelining registers out of *x* levels of adders in the tree.

Pipelining	30 × 30					
	2-Bit coef. 1-bit data			8-Bit coef. 8-bit data		
	1:1	1:2	1:3	1:1	1:2	1:3
Flip-flops	1599	1023	831	26,784	14,370	10,140
T_{min} (ns)	4.65	6.41	8.19	5.66	7.73	9.80

7. Conclusions

A solution to the problem of computing 2-D convolution with medium-large kernels in real time is presented in this paper. It is an FPGA-based architecture that allows to achieve high performance using low clock frequencies thanks to exploiting parallelism at different levels. The design, based on Distributed Arithmetic, uses the FPGA general purpose resources to implement the operations involved in the computation. Our approach is modular and coefficient independent, so it can be customized for any kernel size and for any data or coefficient resolutions. Flexibility, through the adjustment of the pipelining depth and the degree of parallelism at word-level, also permits to reach a trade-off between the application-specific processing data rate and the available resources in a particular FPGA device. Additionally, as a novel contribution, a hardware controller for the proper and efficient treatment of operations at the borders of the input array has been designed. The timing results reported reveal the high throughput of the architecture, capable of processing high-definition video frame rate in real-time. It is also noteworthy that the increasing of both the kernel size and the data resolution do not compromise the performance. We consider that all of these

Table 2
Summary of occupied resources and timing for the implementation on Virtex-4 of different kernels with 8-bit data and 8-bit coefficients.

	Serial			Parallel		
	20 × 20	30 × 30	50 × 50	20 × 20	30 × 30	50 × 50
Flip-flops	8359	18,408	51,356	13,089	26,784	74,298
LUTs 4-input	6045	12,696	35,284	18,622	39,729	124,687
Area (*slices*)	5498	12,068	33,639	13,086	28,731	74,340
T_{min} (ns)	5.10	5.39	6.34	5.64	5.66	6.72

Table 4
Review of implementations found in the literature. 18-bit is assumed for data and coefficients in designs using embedded multipliers unless other resolutions are explicitly indicated in the paper.

	Kernel size	Bits data	Bits coef.	Ms/s	Platform
Perri et al. [16]	5×5	16	16	28	XCV2000e
Sankaradas et al. [3]	5×5	16	18	115	V5LX330T
Wong et al. [36]	7×5	8	18	200	XC2V6000
Bouganis et al. [27]†	7×7	18	18	–	–
Torres et al. [15]‡	7×7	8	8	60	XCV2000e
Porter et al. [17]	7×7	8	8	133	XC2VP100
Farabet et al. [5]	7×7	8	16	200	XCV4SX35
Bouganis et al. [24]	9×9	18	18	91	XC2V6000-4
Zhang et al. [25]‡†	10×10	10	10	69	XC2V2000-4
Zhang et al. [25]‡†	12×12	10	10	67	XC2V2000-4
Fons et al. [18]	13×13	8	18	50	XCV4LX25
Xia et al. [23]	15×15	8	8	–	XC3S1000
Bouganis et al. [24]	15×15	18	18	–	–
Zhang et al. [25]‡†	16×16	10	10	67	XC2V2000-4
Bouganis et al. [35]	21×21	–	10	–	XC2V8000
Zhang et al. [25]‡†	22×22	10	10	66	XC2V2000-4
Embedded mults‡‡	7×7	8	16	183	XC4VLX160-10
	11×11	8	8	142	XC4VFX140-10
	22×22	8	8	149	XC4VSX55-10
Toledo et al.	7×7	8	16	175	XCV4LX160-10
	11×11	8	8	181	XCV4LX160-10
	22×22	8	8	113	XC2V4000-4
	22×22	8	8	177	XC4VLX160-10
	30×30	8	16	171	XCV4LX160-10

'–' Value not reported.
† Filters approximation to reduce operations using lossy algorithm.
‡ Timing reported by Synthesis tool, not *Place*&*Route*.
‡† Limited to quadrant symmetric kernels. 10-Bit data in logarithmic.
‡‡ 22×22 is the largest square-window kernel that can be implemented in the XC4VSX55, the device with more DSP slices (512) in the Virtex-4 family.

features make the architecture of interest for its inclusion in embedded systems and portable applications.

Our proposal has been compared to a number of recent works found in the literature. The results of the comparison evidence the validity of the architecture. In spite of its interest in many fields of applications, no other work addressing the problem of the real-time implementation of such large 2-D convolvers has been found. For similar kernel sizes, our proposal outperforms other works for kernel sizes from 9×9 and larger.

The memories where the sub-convolutions are carried out have been described as ROM memories in the present paper. However, Look Up Tables are actually formed from SRAM cells. Therefore, LUTs can be also deployed to define RAM memories whose contents can be modified on the fly, which entails that time-variant coefficients can be considered for the kernels without the need of reprogramming the FPGA dynamically. The design of a high-level module that automatically fixes the memory contents for any set of coefficients and manages to save them in the memories is the main future work.

Acknowledgment

This work has been partially supported by the Fundación Séneca de la Región de Murcia through the Research Project 15419/PI/10 and by the Ministerio de Economía y Competitividad of Spain through the Research Project AYA2011-14245-E.

References

[1] R.C. González, R.E. Woods, Digital Image Processing, Addison-Wesley (2007).
[2] R. Jain, R. Kasturi, B. Schunck, Machine Vision, McGraw-Hill (1995).
[3] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, H.P. Graf, A massively parallel coprocessor for convolutional neural networks, in: IEEE Int. Conf. on Application-specific Systems, Architectures and Processors ASAP, 2009, pp 53–60.
[4] V. Sriram, D. Cox, K.H. Tsoi, W. Luk, Towards an embedded biologically-inspired machine vision processor, in: Int. Conf. on Field-Programmable Technology FPT, 2010, pp. 273–278.
[5] C. Farabet, C. Poulet, J.Y. Han, Y. LeCun, CNP: an FPGA-based processor for convolutional networks, in: Int. Conf. on Field Programmable Logic and Applications FPL, 2009, pp. 32–37.
[6] J.J. Martínez, F.J. Toledo, E. Fernández, J.M. Ferrández, A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing, Neurocomputing 71 (4) (2008) 766–775.
[7] J.J. Martínez, F.J. Toledo-Moreo, E. Fernández, J.M. Ferrández, Study of the contrast processing in the early visual system using a neuromorphic retinal architecture, Neurocomputing 72 (4–6) (2009) 928–935.
[8] S. Gunn, On the discrete representation of the Laplacian of Gaussian, Pattern Recognition 32 (1999) 1463–1472.
[9] P. Veelaert, K. Teelen, Adaptive and optimal difference operators in image processing, Pattern Recognition 42 (2009) 2317–2326.
[10] M. Nixon, A. Aguado, Feature Extraction and Image Processing, Newnes, 2008.
[11] I.N. Bankman, Handbook of Medical Imaging: Processing and Analysis Management, Elsevier Academic Press (2000).
[12] J. Starck, F. Murtagh, Handbook of Astronomical Data Analysis, Elsevier (2002).
[13] M. Gokhale, P.S. Graham, Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays, Springer (2005).
[14] F. Cardells-Tormo, P. Molinet, Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing, IEEE Transactions on Circuits and Systems 53 (2) (2006) 105–109.
[15] C. Torres-Huitzil, M. Arias-Estrada, FPGA-based configurable systolic architecture for window-based image processing, EURASIP Journal on Applied Signal Processing (2005) 1024–1034.
[16] S. Perri, M. Lanuzza, P. Corsonello, G. Cocorullo, A high-performance fully reconfigurable FPGA-based 2D convolution processor, Microprocessor and Microsystems 29 (8–9) (2005) 381–391.
[17] R.B. Porter, J.R. Frigo, M. Gokhale, C. Wolinski, F. Charot, C. Wagner, A run-time reconfigurable parametric architecture for local neighborhood image processing, in: Euromicro Conference on Digital System Design, 2006, pp. 107–115.
[18] F. Fons, M. Fons, E. Cantó, Run-time self-reconfigurable 2D convolver for adaptive image processing, Microelectronics Journal 42 (2011) 204–217.
[19] J.A. Kalomiros, J. Lygouras, Design and evaluation of a hardware/software FPGA-based system for fast image processing, Microprocessor and Microsystems 32 (2008) 95–106.
[20] J. Mori, C. Sánchez-Ferreira, D.M. Muoz, C.H. Llanos, P. Berger, An unified approach for convolution-based image filtering on reconfigurable systems, in: Southern Conf. on Programmable Logic SPL, 2011, pp. 63–68.
[21] B. Bosi, G. Bois, Y. Savaria, Reconfigurable pipelined 2-D convolvers for fast digital signal processing, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 7 (3) (1999) 299–308.
[22] X. Liang, Mapping of generalized template matching onto reconfigurable computers, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 11 (3) (2003) 485–498.
[23] H. Zhang, M. Xia, G. Hu, A multiwindow partial buffering scheme for FPGA-based 2-D convolvers, IEEE Transactions on Circuits and Systems 54 (2) (2007) 200–204.
[24] C.S. Bouganis, S.B. Park, G.A. Constantinides, P.Y.K. Cheung, Synthesis and optimization of 2D filter designs for heterogeneous FPGAs, ACM Transactions on Reconfigurable Technology and Systems 1 (4) (2009) 24-1–24-28.
[25] M. Zhang, V. Asari, An efficient multiplier-less architecture for 2-D convolution with quadrant symmetric kernels, Integration the VLSI Journal 40 (2007) 490–502.
[26] K. Benkrid, S. Belkacemi, Design and implementation of a 2D convolution core for video applications on FPGAs, in: IEEE Int. Workshop on Digital and Computational Video, 2002, pp. 85–92.
[27] C.S. Bouganis, P.Y.K. Cheung, G.A. Constantinides, Heterogenity exploration for multiple 2D filter designs, in: Int. Conf. on Field Programmable Logic and Applications FPL, 2005, pp. 263–268.
[28] J. Toledo, J. Martínez, J. Garrigós, J.M. Ferrández, Arquitectura para el cómputo de la convolución 2D con plantillas de gran tamaño en tiempo real, in: Jornadas Computación Reconfigurable y Aplicaciones (JCRA), 2011, pp. 251–258.
[29] C.S. Burrus, Digital filter structures described by distributed arithmetic, IEEE Transactions on Circuits and Systems CAS-24 (12) (1977) 674–680.
[30] S. White, Applications of distributed arithmetic to digital signal proccesing: a tutorial review, IEEE Acoustics, Speech and Signal Processing Magazine 6 (3) (1989) 4–19.

[31] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, third ed., Springer, 2007.
[32] B. Kisacanin, S. Bhattacharyya, S. Chai, Embedded Computer Vision, Springer (2009).
[33] F. Javier Toledo-Moreo, J. Javier Martinez-Alvarez, J. Manuel Ferrandez-Vicente, Hand-based interface for augmented reality, in: Int. Symp. on Field-Programmable Custom Computing (FCCM), 2007, pp. 291–292.
[34] B. Cope, P. Cheung, W. Luk, L. Howes, Performance comparison of graphics processors to reconfigurable logic: a case study, IEEE Transactions on Computers 59 (4) (2010) 433–448.
[35] C.S. Bouganis, P.Y.K. Cheung, G.A. Constantinides, A novel 2D filter design methodology for heterogeneous devices, in: Int. Conf. on Field Custom Computing Machines (FCCM), 2005, pp. 13–22.
[36] S.C. Wong, M. Jasiunas, D. Kearney, Fast 2D convolution using reconfigurable hardware, in: Int. Symp. on Signal Processing and its Applications, 2005, pp. 791–794.
[37] K. Pauwels, M. Tomasi, J. Díaz, E. Ros, M. Van Hulle, A comparison of FPGA and GPU for real time phase-based optical flow, stereo and local image features, IEEE Transactions on Computers 61 (7) (2012) 999–1012.
[38] C.W. Fletcher, I. Lebedev, N.B. Asadi, D.R. Burke, J. Wawrzynek, Bridging the GPGU–FPGA efficiency gap, in: ACM/SIGDA Int. Symp. on Field Programmable Gate arrays, 2011, pp. 119–122.

F. Javier Toledo Moreo received the B.Sc. and M.Sc. degrees in electrical and electronic engineering from the Universidad de Murcia, Spain, in 1995 and 1999, respectively. He is currently working toward the Ph.D. degree in computer science at the Universidad Politécnica de Cartagena, Spain. Since 2001 he has been an Assistant Lecturer with the Departamento de Electrónica y Tecnología de Computadoras, Universidad Politécnica de Cartagena. His research interests include signal and image processing, reconfigurable computing and augmented reality.

J. Javier Martínez Alvarez received the B.Sc. and M.Sc. degrees in electrical and electronic engineering form the Universidad Politécnica de Cartagena, Spain, in 1997 and 2001, respectively. He is currently pursuing his Ph.D. in computer science at the same university. Since 2001 he has been an Assistant Lecturer with the Departamento de Electrónica y Tecnología de Computadoras, Universidad Politécnica de Cartagena. His research interests range from signal and image processing, reconfigurable computing to computational neuroscience and biologically accurate modeling.

Javier Garrigós Guerrero received the B.Sc. and M.Sc. degrees in electrical engineering from the Universidad de Murcia, Murcia, Spain, in 1992 and 1995 respectively, and the PhD degree from the Universidad Politécnica de Cartagena, Cartagena, Spain, in 2002. From 1995 to 1996 he was with the Universidad de Zaragoza, as an Invited Researcher. From 1997 to 2002 he was an Assistant Professor with the Departamento de Ingeniería y Tecnología de Computadoras, Universidad de Murcia, and later at the Universidad Politécnica de Cartagena, where he is currently an Assistant Professor with the Departamento de Electrónica, Tecnología de Computadoras y Proyectos. His research interests are in the fields of parallel, reconfigurable and application-specific processing architectures, mainly when oriented to soft-computing (FIS, ANNs and evolutionary computation) and bio-inspired systems.

J. Manuel Ferrández Vicente received the B.Sc. degree in Computer Science in 1995, and the Ph.D. degree in 1998, all of them from the Universidad Politécnica de Madrid, Spain. He is currently Associate Professor at the Department of Electronics, Computer Technology and Projects at the Universidad Politécnica de Cartagena and Director of the Electronic Design and Signal Processing Research Group at the same University. His research interests include bioinspired processing, neuromorphic engineering and augmented reality systems. He is actively working on the development of a visual prosthesis for visually impaired people.