computers

MDPI

Article

Design of a Convolutional Two-Dimensional Filter in FPGA for Image Processing Applications

Gian Domenico Licciardo *, Carmine Cappetta and Luigi Di Benedetto

Department of Industrial Engineering, University of Salerno, 84084 Fisciano (SA), Italy; ccappetta@unisa.it (C.C.); ldibenedetto@unisa.it (L.D.B.)

* Correspondence: gdlicciardo@unisa.it; Tel.: +39-89-96-4303

Academic Editor: Laith Al-Jobouri

Received: 14 April 2017; Accepted: 15 May 2017; Published: 17 May 2017

Abstract: Exploiting the Bachet weight decomposition theorem, a new two-dimensional filter is designed. The filter can be adapted to different multimedia applications, but in this work it is specifically targeted to image processing applications. The method allows emulating standard 32 bit floating point multipliers using a chain of fixed point adders and a logic unit to manage the exponent, in order to obtain IEEE-754 compliant results. The proposed design allows more compact implementation of a floating point filtering architecture when a fixed set of coefficients and a fixed range of input values are used. The elaboration of the data proceeds in raster-scan order and is capable of directly processing the data coming from the acquisition source thanks to a careful organization of the memories, avoiding the implementation of frame buffers or any aligning circuitry. The proposed architecture shows state-of-the-art performances in terms of critical path delay, obtaining a critical path delay of 4.7 ns when implemented on a Xilinx Virtex 7 FPGA board.

Keywords: visual search; multiplier; 2D filtering modules; Gaussian filtering

---

1. Introduction

In image processing applications such as object recognition, segmentation or Visual Search (VS), a filtering stage is always required. In particular, often the convolution operation has to be performed directly in two dimensions, due to the inherently two-dimensional (2D) nature of the object to process. Moreover, during the last years the computational complexity of designs dedicated to this kind of problems underwent a critical growth, due to more complex architecture designs and an increase in the amount of data to be processed, related to higher resolution images. These kind of issues are typical of any algorithm working on 2D portions of images [1,2] and they are relevant in both software (SW) and hardware (HW) applications.

In particular, VS search applications are among those which require most in terms of complexity, due to the high amount of calculations needed for searching and extracting the features in the input frame [3]. However, in these kind of applications, SW implementations are usually limited in terms of maximum achievable frequency, which does not allow for real-time processing in the case of higher resolution images unless using strong simplifications of the algorithms. Among the most important 2D filtering applications, one of the most important regards the convolution between a tile of the image and several Gaussian kernels to obtain the Difference-of-Gaussian (DoG) scale-space pyramid [4]. These elaborations are required in the Scale Invariant Feature Transform (SIFT) [3,4] algorithm, for example, where they represent the most computationally demanding step. In this case, SW implementations of the algorithm require long elaboration times to process a single image, thus they are not capable of achieving real-time performances, while the problem is worse for images having higher resolutions [4–8]. For these reasons, HW implementations of filtering stages are more and more frequent, but it has to be underlined that the effort to obtain optimized designs is not justified by the real-time requirements only. In fact, the diffusion of handheld devices and Internet-of-Things (IoT) applications is leading the research to new designs, capable of achieving better Area-Power-Delay (ADP) trade-offs. Starting from these considerations, during the last years, several HW design have been proposed aiming to obtain optimized filtering architectures dedicated to SIFT and feature extraction [8–10].

Another issue arising in applications dealing with image processing is related to the large amount of data to be processed and stored and thus to the memory requirements for frame buffering. At this end, other works have focused on memory management in order to develop architectures capable of optimizing memory designs, together with writing and reading operations. The solution proposed in [11] exploits custom coding and a partial serialization of the filtering operation to obtain a buffer-less solution. The design is tailored for DoG operations and its method could not be used as presented for other VS applications or other methods not using Gaussian kernels. On the contrary, the design presented in [12] could be used with kernels different from the Gaussian one, since it does not recur to the separability property of the Gaussian kernels, presenting a complex arrangement of SRAMs modules, row shuffling operations and dedicated logic, which complexity grows with the dimensions of the tile to be processed.

Another huge difference between SW and HW implementations regards the simplification of the operands from IEEE-754 floating point data (FP32) to fixed point (FI) ones. This simplification reduces the accuracy of the design to obtain units capable of performing multiply-accumulate (MAC) operations with better ADP performances, while reaching a trade-off on the accuracy of performances. Meanwhile, very few studies have been conducted to optimize the arithmetic units and improve their performances. Other simplifications of the above-cited methods regard reductions of the algorithm's complexity, such as the reduction in the number of scales or octaves or a decrease of the value of the standard deviation of the Gaussian filter in the case of SIFT applications [13,14].

The aim of the present work is to develop a novel 2D convolutional filter for the processing of images obtaining FP32 results, showing low area occupation and power consumption, while achieving good speed performances. This has been achieved exploiting an ad hoc partitioning method of the operands [15,16], dedicated to multimedia applications and to all other applications which present a well-defined integer input range and working on kernels of fixed coefficients. Moreover, the design is capable of working on continuous data streams, which could be provided by either an external memory or an acquisition device, such as an image sensor without using any frame buffer, thanks to a careful organization and management of small intermediate buffers, which allow for on-the-fly calculations. The organization of the data and the management of the memory structures have been favored by the optimized hard macros available in modern Field Programmable Gate Arrays (FPGAs) to implement Block-RAM (BRAM) modules. However, the FPGA implementation has been chosen not only for the availability of hard macros for the memory structures, but also for prototyping purposes, before developing an Application Specific Integrated Circuit (ASIC) design. Avoiding the use of multipliers, the design achieves great area reduction if compared to structures having the same accuracy, allowing to greatly improving the performances to obtain a FP32 compliant result in this case. The implementation of the proposed design on a high-end FPGA board shows a worst delay path of 4.7 ns to elaborate an IEEE-754 FP32 compliant results in the slow/slow corner, considering a 3×3 pixels window as input.

2. Mathematical Background

The implemented partitioning method derives from a problem stated for the first time by the French mathematician Claude-Gaspard Bachet de Méziriac in the XVII century. The original formulation of the problem regarded how to establish the minimum set of integers to obtain any integer number included in the range [1; 40] using addition and subtraction of the parts composing the set. The problem, known as Bachet's Weights Problem, concerns theory of numbers and was analytically discussed for the first time by Hardy and Wright in [17]. However, a solution to the generalized Bachet's problem, in which 40 could be replaced with any integer number, has been developed in the last years in [18,19]. A complete review on the problem, its discussion and on the relative theorems is provided in [20]. The principal conclusions which are useful in understanding the proposed design are summed up in the following.

Theorem 1. *Given a range of integers [0; r], it is possible to define a set of integers* $S_r = \{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n\}$ of cardinality $n + 1 = \lfloor \log_3(2r) \rfloor + 1$ *capable of obtaining all the values in the given range as a combination of the terms* $\lambda_i \in S_r$, called parts.

Theorem 2. *The set* S_r *is composed by union of the first n powers of 3 and the term* $R = r - (3^0 + 3^1 + 3^2 + \ldots + 3^{n-1})$*. The obtained set could then be rewritten as* $S_r = \{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n\} := \{3^0, 3^1, 3^2, \ldots, 3^{n-1}, R\}$.

Theorem 3. *The obtained partition is a unique one and does not exist any other partition, S_t, of the set [0; r] which satisfies the conditions stated in Theorem 1 and Theorem 2 and is composed by fewer than n+1 elements, as S_r.*

The reported theorems allow to express any integer number in the input range, $q \in [0;\ r]$, as a combination of the previously calculated parts

$$q = \sum_{i=0}^{n} C_i \lambda_i \tag{1}$$

where C_i is the generic coefficient defined in the set $C := \{-1, 0, 1\}$ and used to obtain the correct choice of the parts. The application of the proposed method to obtain any q number is given in Table 1, in the general case of $(n+1)$ parts and.

Table 1. Application of the proposed method.

Input Value	**Bachet Partition**					
	3^0	3^1	3^2	3^3	...	λ_n
0	0	0	0	0	...	0
1	+1	0	0	0	...	0
2	−1	+1	0	0	...	0
3	0	+1	0	0	...	0
4	+1	+1	0	0	...	0
5	−1	−1	+1	0	...	0
...	...	...	...	...	...	...
q	C_0	C_1	C_2	C_3	...	C_n
...	...	...	...	...	...	...
r	+1	+1	+1	+1	...	+1

Equation (2) shows an example in the case of an 8 bits input, in which the partition set is defined as $S_r = \{1, 3, 9, 27, 81, 134\}$ for the input 42

$$q = 42 = (0) \times 1 + (-1) \times 3 + (-1) \times 9 + (-1) \times 27 + (1) \times 81 + (0) \times 134 \tag{2}$$

and the set of values $C_i = \{0; -1; -1; -1; +1; 0\}$.

It is now possible to understand how Equation (1) is capable of obtaining a partition of a generic multiplication between an input pixel, representing an integer value, and a value taken from a set and representing a coefficient of a Gaussian kernel. Doing this operation also for the other pixels composing the input tile, having dimensions $K \times K$, it is then possible to obtain the result of the 2D convolution between the input tile and a generic kernel F, of the same dimensions. The result of the filtering operation for the central pixel of the tile $I(x_0, y_0)$ could be rewritten as

$$\begin{aligned} O(x_0,y_0) &= \sum_{h=0}^{K-1}\sum_{j=0}^{K-1} F(h,j)I\left(x_0+h-\tfrac{K-1}{2}, y_0+j-\tfrac{K-1}{2}\right) = \\ &= \sum_{h=0}^{K-1}\sum_{j=0}^{K-1} F_{h,j}\sum_{i=0}^{n} C_i\left(x_0+h-\tfrac{K-1}{2}, y_0+j-\tfrac{K-1}{2}\right)\lambda_i = \sum_{h=0}^{K-1}\sum_{j=0}^{K-1}\sum_{i=0}^{n}\left(F_{h,j}\lambda_i\right)C_i\left(x_0+h-\tfrac{K-1}{2}, y_0+j-\tfrac{K-1}{2}\right) \end{aligned} \tag{3}$$

where $F(h,j) = F_{h,j}$ and the general input tile $I(x,y)$ is a matrix in which each element can be decomposed as in (1).

In typical image and video applications working on gray scale images, such as in the case of VS, the input data is coded using 8 bits, representing an integer number in the range $[0;\ 2^8-1]$. In these cases, choosing the dimensions of the kernel, it is possible to obtain the coefficients of the filter, F_j, and precompute all the partial products between all the $F_{h,j}$ and all the parts λ_i. All these pre-calculated products could then be used to rewrite the convolution operation as a summation of the various terms, P:

$$O(x_0,y_0) = \sum_{h=0}^{K-1}\sum_{j=0}^{K-1} P\left(x_0+h-\frac{K-1}{2}, y_0+j-\frac{K-1}{2}\right) \tag{4}$$

where

$$P\left(x_0+h-\frac{K-1}{2}, y_0+j-\frac{K-1}{2}\right) = F(h,j)I\left(x_0+h-\frac{K-1}{2}, y_0+j-\frac{K-1}{2}\right).$$

The minimality property of the S_r set, obtained using the proposed partition method, guarantees a simplified computational complexity to implement Equation (4), if compared to a classical base-2 Distributed Arithmetic (DA) method. Moreover, compared to a base-2 DA decomposition method, the proposed architecture avoids shift operations.

3. The Proposed Architecture

The proposed method has been used to implement a 2D filter, starting from a single FP32 multiplier unit developed using the Bachet decomposition. The input pixels are, in general, coded on m bits, representing unsigned integer data, and they are acquired in raster-scan mode, without using input frame buffers or caching apparatus, except from the one provided by the input acquisition device itself. The main units composing the architecture are:

- a Coefficient Generator Unit;
- a Memory Module;
- a Filtering Module.

A block scheme of the proposed implementation is depicted in Figure 1.

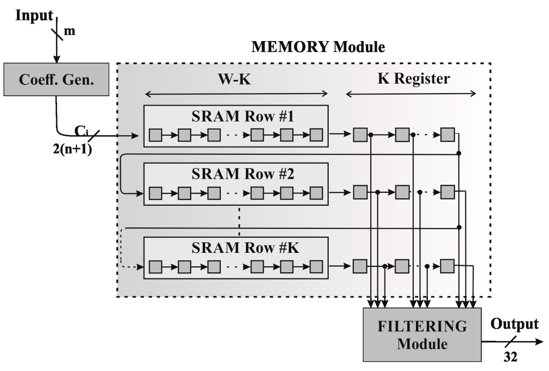

Figure 1. Block scheme of the proposed 2D filtering architecture.

3.1. Coefficient Generator Unit

In order to calculate the combination of the various λ_i parts, a unit capable of generating the various C_i coefficients for any q of the input range is needed. This unit, called Coefficient Generator Unit, takes as input the Uint-m data and returns a $2(\lfloor \log_3(2^{m+1}) \rfloor + 1)$ bits string as output, since every sign coefficient C_i is coded on 2 bits. The unit is principally made up by a Read-Only Memory (ROM) implementing the coding reported in Table 1, where the input range, i.e. the value of the input length in bits, m, establishes the size of the ROM needed to store all the coefficients, which value is

$$M_C = 2^{m+1} \cdot (\lfloor \log_3(2m) \rfloor + 1) \quad (5)$$

Obviously, after this operation the input pixel value is no longer needed, since this information is coded in the associated coefficients that are sent to the Memory Module. Figure 2 reports the memory requirement for the Coefficient Generator ROM varying the number of the input bits.

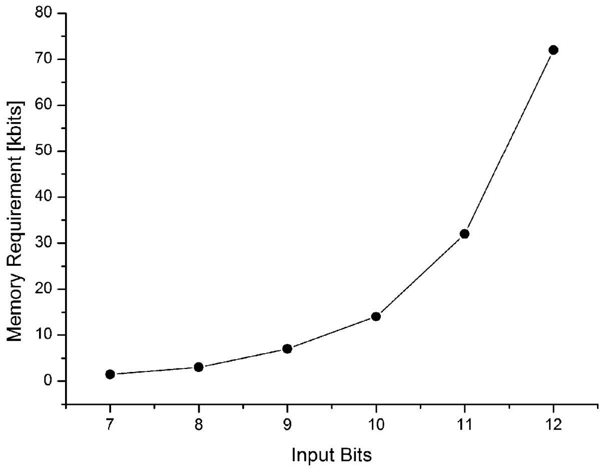

Figure 2. Memory requirements varying the number of input bits.

3.2. Memory Module

The data coming from the Coefficient Generator Unit are stored in a Serial Input Parallel Output (SIPO) unit. The SIPO takes the input pixels in raster scan order and could be represented as a stripe buffer having dimensions $K \cdot W$, to store the first K rows of the frame to be elaborated. Only when the first K values of the last row arrive to the buffer, the data related to the tile to be processed are ready to be sent to the filtering stage. In this way, after the Memory Module stored $(K-1) \times (W-1) + K$ values it is possible to process one pixel per cycle, due to the organization of the memory described above. In fact, the insertion of a new data in the Memory Module, causes the shift of those stored in a way that data to be filtered result naturally aligned in the new tile, without any additional operation.

An implementation of the described Memory Module could be carried out using registers to compose the entire stripe buffer, but this kind of solution has the major drawback of requiring a large amount of physical resources. In fact, for an 8 bit input, in the case of working on VGA frames (640×480 pixels), would require, in the case of a kernel having $K = 25$, $640 \times 25 = 16,000$ values; considering that each value is coded on 12 bits, the total amount of memory to be implemented has to store 188 Kbits. A much more fitting solution consists of using embedded BRAMs modules, emulating the SIPO structure behavior, to store a complete frame row, with the exception of the data to be sent to the Filtering Module which are stored in registers instead. On the contrary, the $K \cdot K$ tile has to be stored in registers, in order to preserve the data stream and to allow the transfer of all the tile to the final filtering structure. It is important to note that, due to the use of RAM structures, the shifting operations do not involve any data transfer, but only a careful management of the generation of the addresses to emulate the shift operations and maintain the correct alignment of the data. The solution is also advantageous due to the availability of hard macros in modern FPGA boards, which are optimized for data transferring and storing operations. Using a partial tile buffer, the proposed implementation is always better in terms of resource requirements if compared to a buffer-based one in terms of amount of memory to be stored.

3.3. Filtering Module

In order to calculate (3), the pre-calculated products $F_{h,j}\lambda_i$ are stored in $(n+1)$ small ROMs, one for each part, whose dimensions are reduced to $\sum_{i=1}^{\lceil K/2 \rceil} i$ words for the symmetry of the kernel. A further simplification of the operation to be performed could be achieved storing also the 2's complements of the pre-multiplied coefficients in the ROMs. In this way, when the generic $C_i = -1$, any further logic circuitry to complement the operands for the subtraction operation is not needed. The outputs of the ROMs together with the coefficients bits coming from the Memory Module are sent to the Bachet multiplier units. It is important to note that all the ROMs must be read at the same time. The structure of the developed equivalent FP32 multiplier unit is depicted in Figure 3.

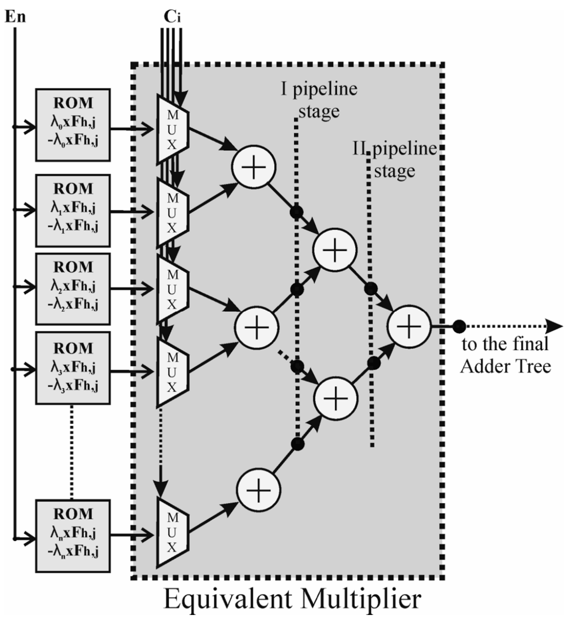

Figure 3. Block diagram of the Bachet multiplier unit.

It is basically composed of an adder tree structure, having depth $\lceil \log_2(n+1) \rceil$ and made up by n adder units. The operation performed by the architecture is the one presented in Equation (3): the pre-multiplied coefficients are selected using a multiplexer and the C_i coefficients coming from the stripe buffer as selection inputs. The partial results of the multiplications then have to be added together with the products calculated for the other pixels, to complete the convolution operation.

Moreover, the operands have been custom recoded to obtain further simplifications of the complexity of the proposed architecture, without accuracy losses. In fact, starting from considerations of the FP32 format it is possible to code the starting coefficients and the partial results as FI data, establishing the appropriate codelength in order to not undermine the accuracy of the result of the single multiplication and of the overall convolution operation. To obtain this implementation, first a careful study of the IEEE-754 coding of the product between the kernel coefficients and the parts set has to be conducted. Then, all the exponents of the pre-multiplied coefficients have to be reported to the greatest exponent among all the used coefficients and the significands have to be shifted according to the difference between the maximum exponent value and the significand exponent. Hence, to obtain the correct FI coding, the codelength of each operand has to be increased to avoid any truncation of the operands that could undermine the accuracy of the result. Given the minimum and maximum kernel coefficients, $F^{\min}$ and $F^{\max}$, it is possible to obtain an analytical formula to choose the adequate codelength, l_S, as

$$l_S = \left\lceil 23 + \log_2\left(\frac{F^{\max}\lambda_n}{F^{\min}\lambda_0}\right)\right\rceil \tag{6}$$

where 23 bits are the length of the standard FP32 significand while λ_0 and λ_n represents the minimum and maximum element of the calculated part set, respectively. Using this type of coding, the mantissas of the partial results do not need any normalization operation and the exponents are not taken into account in these operations, causing a resource and thus an area saving. The normalization operations on both the mantissa and exponent of the final results will be then performed in order to obtain a FP32 compliant format, allowing to correctly reuse the outputs of the Filtering Module for further elaborations.

3.4. Memory Reduced Bachet Multiplier Architecture

A study on the amount of memory to be stored, increasing the dimensions of the tile to be processed, has been conducted. The results are shown in Figure 4.

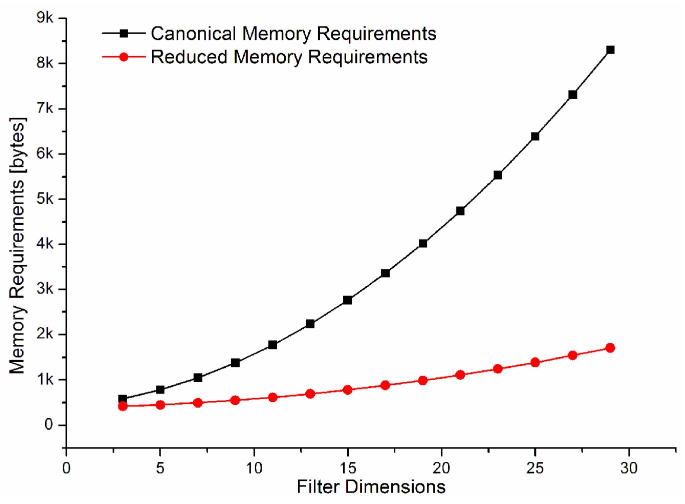

Figure 4. Required memory resources of the 2D convolution-based filter as a function of its dimensions, when m = 8 bits.

Given an input range, the amount of memory to be stored related to the C_i does not increase with the filter, as showed in Equation (5). On the contrary, the number of bits to be stored that are dedicated to the pre-multiplied coefficients undergoes a huge increase with the filter size. Analytically the number of bits, M_{parts}, increases as

$$M_{parts} = 2l_S(n+1)\sum_{i=1}^{\lceil K/2\rceil} i \tag{7}$$

where l_S is the coefficient codelength, $(n+1)$ is the number of parts and $\sum_{i=1}^{\lceil K/2\rceil} i$ is the number of coefficients to be considered.

This could result in unacceptable memory requirements in the case of memory-constrained applications. We developed a memory-reduced implementation of the Bachet multiplier, which is capable of obtaining the other $F_{h,j}\lambda_i$ coefficients starting from $F_{h,j}\lambda_0$ and their 2's complement counterparts. The other coefficients are obtained by shift and add operations on the stored $F_{h,j}\lambda_0$ and

$-F_{h,j}\lambda_0$ stored terms. Table 2 reports the shift and add operations in the case of an 8 bit input. In this case, Equation (6) could be rewritten as

$$M_{parts} = 2l_S \sum_{i=1}^{\lceil K/2 \rceil} i \tag{8}$$

Table 2. Detailed shift and add operations for an 8 bits input design.

	Operation	# of Shifts	# of Additions
$\lambda_0 = 1$	–	0	0
$\lambda_1 = 3$	$2 + 1$	1	1
$\lambda_2 = 9$	$2^3 + 1$	3	1
$\lambda_3 = 27$	$2^5 - 2^2 - 1$	7	2
$\lambda_4 = 81$	$2^6 + 2^4 + 1$	9	2
$\lambda_5 = 134$	$2^7 + 2^2 + 2$	10	2
$-\lambda_0 = -1$	–	0	0
$-\lambda_1 = -3$	$-2 - 1$	1	1
$-\lambda_2 = -9$	$-2^3 - 1$	3	1
$-\lambda_3 = -27$	$-2^5 + 2^2 + 1$	7	2

This approach reduces the number of embedded ROMs to be implemented on the targeted FPGA at the cost of allocating more logic and algebraic elements. We have to underline that in the 8 bit input case there is no need to store the 2's complements of λ_4 and λ_5, since the related pre-multiplied coefficients are never used in the decomposition operations for the given input range.

4. Results

At the end of obtaining an implementation that could be easily compared to the ones presented in literature related to VS applications, we decided to implement a 2D Gaussian kernel, working on 8 bit inputs. In typical VS applications a minimum size $K = 3$ is fixed for the kernel, thus this size has been chosen for the Gaussian kernel. The input range fixes the r parameter, $r = 2^m - 1 = 255$, and thus from Theorem 1 and Theorem 2 it is possible to state that $n + 1 = \lfloor \log_3(2r) \rfloor + 1 = 6$ parts are needed, while the parts set is given by $S_r = \{1, 3, 9, 27, 81, 134\}$ and the codelength requirement to code the sign coefficients C_i for each input pixel is $l_c = 12$ bits.

In order to obtain the codelength on which to code the pre-calculated coefficients, further considerations have to be developed on the filter. A standard 2D symmetric Gaussian kernel function is analytically represented by

$$G(x, y, \sigma) = (2\pi)^{-1}\sigma^{-2}e^{-\frac{x^2+y^2}{4\sigma^2}} \tag{9}$$

and starting from that it is possible to obtain the $F^{\min}$ and $F^{\max}$ values

$$\begin{aligned} F^{\min} = G^{\min} = G(3, 3, \sigma) = (2\pi)^{-1}\sigma^{-2}e^{-\frac{18\sigma^2}{2\sigma^2}} = (2\pi)^{-1}\sigma^{-2}e^{-9} \\ F^{\max} = G^{\max} = G(0, 0, \sigma) = (2\pi)^{-1}\sigma^{-2} \end{aligned} \tag{10}$$

Since $\lambda_0 = 1$ and $\lambda_n = 134$, substituting these values in (5) we obtain a length of the significands $l_S = \lceil 23 + \log_2(\lambda_n e^9) \rceil = 44$ bits. An example of the custom recoding is given in Table 3.

Table 3. Detailed shift and add operations for an 8 bits input design.

Smallest Coeff.	r	λ_n	FP32 Coding of 1.1×10^{-3}	Modified Coding of 1.1×10^{-3}
1.1×10^{-3}	255	134	0 01110101001 00000010110111100000	0000000000000100100000010110111100000

The results related to the implementation of the proposed filter are instead presented in Table 4, together with a comparison with architectures using conventional FP32 multipliers and a one using Modified Booth (MB) multipliers, used to compare the proposed design with DA related techniques.

Table 4. Synthesis of results for the proposed filter implementation compared with other designs.

	Proposed	Proposed Reduced Memory	Conventional FP32	MB	Prop. FP16	Cabello [21]
Tech.	XC7V	XC7V	XC7V	XC7V	Spartan 6	Spartan 6
LUTs	4750	7760	7732	8606	2395	5052
Mem. [bytes]	582	417	–	528	4 BRAM	1 BRAM
M_C [bytes]	384	384	–		–	–
M_{parts} [bytes]	198	33	–		–	–
Delay [ns]	4.700	4.720	17.432	8.785	6.90	10
Power * [W]	0.684	0.728	0.907	1.226	–	–

* Normalized at 100 MHz.

All the implementations have been targeted to the same FPGA board, considering the multiplication between a FP32 and an 8 bits integer data at the end of a fair comparison. Moreover, to obtain results which are as reproducible as possible, Xilinx IPs have been used in the comparisons to implement the multiplier and adder units. In particular, the multiplier units have been implemented using a 3-stage pipeline design. We imposed not too aggressive constraints, always to obtain a fair comparison between the various designs. With this in mind, we forced the system to use BRAM modules and we do not allow the use of Digital Signal Processors (DSP), aiming to obtain a good estimation for future std_cell implementation, towards an ASIC realization of the proposed design.

As it is possible to notice from the data reported in Table 4, the proposed design is very fit to be implemented on an FPGA board, since it is possible to obtain good speed and area performances exploiting the availability of hard macros to implement both BRAMs and ROMs. The structure developed is capable of processing one pixel per cycle in 4.700 ns. This represents a 371% increase in speed compared to a conventional FP32 multiplier and a 187% increase compared to a MB multiplier. A Floating-Point (FP) implementation on 16 bits of the proposed design has been also developed at the end of comparing the proposed architecture with the one presented in [21]. This design has been implemented on a Xilinx Spartan 6 board, using Nexys 3, in order to obtain a fair comparison with the design in [21]. The results are also reported in Table 4. The number of LUTs required for the proposed implementation are approximately 38.6% and 44.8% lower than in the conventional FP32 and MB cases, respectively. However, in the case of memory-constrained applications, it is possible to use the Memory Reduced implementation of the Bachet multiplier described in Section 3.4. To briefly recall, this solution reduces the number of stored coefficients and, hence, of the related memory, deriving a large number of coefficients by shift and add operations. For these reasons, it shows a small increase in the number of occupied LUTs with respect to the conventional FP32 filter design, while having approximately the same delay path of the classical Bachet filter. Furthermore, it has to be underlined that, since almost the totality of the LUTs considered in Table 4 are used to implement the multipliers, it is correct to say that increasing the size of the tile, K, the LUTs occupation will increase approximately as K^2, while the memory requirements will scale as shown in Figure 4. The performances for the various architectures in terms of frames-per-second, by varying the resolution of the processed frames are reported in Table 5.

Table 5. Performances of the reported designs in terms of frames-per-second (fps) varying the resolution of the frames.

	Proposed	Proposed Reduced Memory	Conventional FP32	MB	Prop. FP16	Cabello [21]
VGA (640 × 480)	692	689	186	370	471	325
Full-HD (1920 × 1080)	102	102	27	54	69	48
4K UHDTV (3840 × 20160)	25	25	6	13	17	12

Furthermore, VS applications usually require great amounts of memory; for this reason the amount of memory required to implement the proposed architecture does not represent an actual problem in most VS applications. In fact, due to the implementation of frame buffers [22] or partial data storage [23–25], in these designs usually the memory requirements are in the order of Mbits, making the additional memory required to store the coefficients negligible.

Finally, it is important to notice that implementations regarding Gaussian filters are some of the most demanding applications in terms of resource usage. For this reason, it has been employed in the proposed design. However, the proposed design can be very easily adapted to different kernel filters, by simply changing the stored coefficients. This modification, in general, causes a favourable reduction of the amount of mapped memory. Meanwhile, the arithmetic complexity of the convolution operation does not vary, depending only on the number of additions to be performed and thus on the dimensions of the kernels of the filters. For example, in the use of a weighted average filter or a Sobel filter it is possible to store fewer coefficients than in the Gaussian case. This results in a simpler implementation and allows for better performances, even on semi-custom devices [26].

5. Conclusions

This work proposes an HW design for two-dimensional filtering applications for image and video. The results are particularly interesting for feature extraction and VS applications, where issues arise due to the high number of operations and required memory. The proposed architecture strongly intervenes on both of these issues, to improve the throughput and reduce the area occupation of the structures at the same time, in the case of FP32 data elaboration. Moreover, a memory-reduced implementation of the proposed method has been developed to be targeted to memory-constrained applications.

This has been done by exploiting a new partitioning method to simplify the arithmetic units and in particular to substitute multipliers with equivalent units formed by small ROMs, simplified adders. This solution is capable of achieving state-of-the-art performances when implemented on an FPGA board.

Acknowledgments: The research activity has been supported from the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR).

Author Contributions: Gian Domenico Licciardo conceived the design; Gian Domenico Licciardo and Carmine Cappetta developed the architecture; Gian Domenico Licciardo, Carmine Cappetta and Luigi Di Benedetto contributed analysis tools and analyzed the data; Gian Domenico Licciardo, Carmine Cappetta and Luigi Di Benedetto wrote the paper. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, S.L. VLSI implementation of an adaptive edge-enhanced image scalar for real-time multimedia applications. *IEEE Trans. Circuits Syst. Video Technol.* **2013**, *23*, 1510–1522. [CrossRef]
2. Basu, M. Gaussian-Based Edge-Detection Methods—A Survey. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2002**, *32*, 252–260. [CrossRef]
3. Lowe, D.G. Object Recognition from Local Scale-Invariant Features. In Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 20–27 September 1999; Volume 2, pp. 1150–1157.
4. Lowe, D.G. Distinctive image features from scale-invariant key points. *Int. J. Comput. Vis.* **2004**, *60*, 91–110. [CrossRef]
5. Mizuno, K.; Noguchi, H.; He, G.; Terachi, Y.; Kamino, T.; Fujinaga, T.; Izumi, S.; Ariki, Y.; Kawaguchi, H.; Yoshimoto, M. A low power real-time SIFT descriptor generation engine for full HDTV video recognition. *IEICE Trans. Electron.* **2011**, *94*, 448–457. [CrossRef]
6. Licciardo, G.D.; Albanese, L.F. Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms. *IET Image Process.* **2012**, *6*, 301–308. [CrossRef]

7. Albanese, L.F.; Licciardo, G.D. High speed CAVLC encoder suitable for field programmable platforms. In Proceedings of the International Conference on Signals and Electronic Systems, Gliwice, Poland, 7–10 September 2010; pp. 327–330.
8. Jiang, J.; Li, X.; Zhang, G. SIFT Hardware Implementation for Real-Time Image Feature Extraction. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *24*, 1209–1220.
9. Chandrasekhar, V.; Takacs, G.; Chen, D.; Tsai, S.; Grzeszczuk, R.; Girod, B. CHoG: Compressed Histogram of Gradients a Low Bit-Rate Feature Descriptor. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009; pp. 2504–2511.
10. Bay, H.; Tuytelaars, T.; van Gool, L. SURF: Speeded up Robust Features. In Proceedings of the 9th European Conference on Computer Vision, Graz, Austria, 7–13 May 2006; pp. 404–417.
11. Licciardo, G.D.; Boesch, T.; Pau, D.; di Benedetto, L. Frame Buffer-less Stream Processor for Accurate Real-Time Interest Point Detection. *Integr. VLSI J.* **2016**, *54*, 10–23. [CrossRef]
12. Huang, F.C.; Huang, S.Y.; Ker, J.W.; Chen, Y.C. High performance SIFT hardware accelerator for real-time image feature extraction. *IEEE Trans. Circuits Syst. Video Technol.* **2012**, *22*, 340–351. [CrossRef]
13. Borg, N.P.; Debono, C.J.; Zammit-Mangion, D. A Single Octave SIFT Algorithm for Image Feature Extraction in Resource Limited Hardware Systems. In Proceedings of the 2014 IEEE Visual Communications and Image Processing Conference, Valletta, Malta, 7–10 December 2014; pp. 213–216.
14. Kim, E.S.; Lee, H.J. A novel hardware design for SIFT generation with reduced memory requirement. *J. Semicond. Technol. Sci.* **2013**, *13*, 157–169. [CrossRef]
15. Licciardo, G.D.; Cappetta, C.; di Benedetto, L.; Vigliar, M. Weighted Partitioning for Fast Multilpier-less Multiple Constant Convolution Circuit. *IEEE Trans. Circuits Syst. II Express Briefs* **2017**, *64*, 1.
16. Licciardo, G.D.; Cappetta, C.; di Benedetto, L.; Rubino, A.; Liguori, R. Multiplier-less Stream Processor for 2D Filtering in Visual Search Applications. *IEEE Trans. Circuits Syst. Video Technol.* **2016**. [CrossRef]
17. Hardy, G.H.; Wright, E.M. *An Introduction to the Theory of Numbers*, 6th ed.; Oxford University Press: Oxford, UK, 2008.
18. Park, S.K. The r-complete partitions. *Discret. Math.* **1998**, *183*, 293–297. [CrossRef]
19. Rødseth, Ø.J. Enumeration of M-partitions. *Discret. Math.* **2006**, *306*, 694–698. [CrossRef]
20. O#x2019;Shea, E. Bachet's problem: As few weights to weigh them all. *arXiv* **2008**.
21. Cabello, F.; Leon, J.; Iano, Y.; Arthur, R. Implementation of a Fixed-Point 2D Gaussian Filter for Image Processing Based on FPGA. In Proceedings of the Signal Processing: Algorithms, Architectures, Arrangements and Applications (SPA), Poznan, Poland, 23–25 September 2015; pp. 28–33.
22. Chao, W.M.; Chen, L.G. Pyramid architecture for 3840 $\times$ 2160 Quad Full High Definition 30 frames/s video acquisition. *IEEE Trans. Circuits Syst. Video Technol.* **2010**, *20*, 1499–1508. [CrossRef]
23. Vigliar, M.; Licciardo, G.D. Hardware Coprocessor for Stripe-Based Interest Point Detection. U.S. Patent 20,130,301,930, 14 November 2013.
24. Licciardo, G.D.; D'Arienzo, A.; Rubino, A. Stream processor for real-time inverse Tone Mapping of Full-HD images. *IEEE Trans. VLSI Syst.* **2015**, *23*, 2531–2539. [CrossRef]
25. Albanese, L.F.; Licciardo, G.D. An area reduced design of the Context-Adaptive Variable-Length encoder suitable for embedded systems. In Proceedings of the 2010 5th International Symposium on I/V Communications and Mobile Network (ISVC), Rabat, Morocco, 30 September–2 October 2010; pp. 1–4.
26. Licciardo, G.D.; Costagliola, M. An H.264 encoder for real time video processing designed for SPEAr™ customizable system-on-chip family. In Proceedings of the IEEE International Conference on Signal Processing and Communications, Dubai, UAE, 24–27 November 2007; pp. 824–827.

© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).