This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

Multiplier-less Stream Processor for 2D Filtering in Visual Search Applications

Gian Domenico Licciardo, *Member*, *IEEE*, Carmine Cappetta, *Student Member*, *IEEE*, Luigi Di Benedetto, *Member*, *IEEE*, Alfredo Rubino, Rosalba Liguori

***Abstract*—A new 2D convolution-based filter is presented specifically designed to improve Visual Search applications. It exploits a new radix-3 partitioning method of integer numbers, derived from the weight partition theory, which allows substituting multipliers with simplified floating-point adders, working on 32 bits floating point filter coefficients. The memory organization allows elaborating the incoming data in raster scan order, as those directly provided by an acquisition source, without frame buffers and additional aligning circuitry. Compared to the existent literature, build around conventional arithmetic circuitry, the proposed design achieves state-of-the-art performances in the reduction of the mapped physical resources and elaboration velocity, achieving a critical path delay of about 4.5 ns both with a Xilinx Virtex 7 FPGA and CMOS 90nm std_cells.**

***Index Terms*—Visual Search, Interest Point Detection, Field programmable gate arrays, Gaussian filter, Multiplier.**

I. Introduction

TWO dimensional (2D) convolution-based filtering is widely used in Visual Search (VS) applications. The increment of VS functionalities and the growing demand for very high quality multimedia applications have led to the exponential growth of the computational complexity of the dedicated Hardware/Software (HW/SW) systems. On the other hand, the growing usage of such applications in hand-held and portable devices gives rise to incompatible constrains in terms of elaboration speed and number of instantiated physical resources [1]. Although this is an actual problem of all the algorithms that locally work on portions of a frame [2], [3], the huge number of calculations required to extract features from an image for recognizing and classifying its content [4], makes VS applications by far the most demanding ones. In such cases, 2D convolution-based Gaussian filters are largely employed, in order to remove high frenquency noise, as well as to construct a Difference-of-Gaussian (DoG) scale-space pyramid from a number of downsampled, blurred specimens of an input image [5]. A number of HW solutions have been proposed to compensate for the inadequate performances of SW implementations [5]-[7], mainly hindered by the large number of floating points (FP) Multiply-ACcumulation (MAC) operations and the large quantity of memory for frame buffering [8]. However, for the fullfillment of severe constrains, also HW implementations need of a number of simplifications that gave rise to less complex alternatives to Gaussian filtering [9], [10]. With reference to the recent literature, we observed that the optimized HW solutions have been essentially addressed to the improvement of memory architectures and the search for the optimal strategy for exchanging data with the datapath circuitries. The bufferless solution in [11] preserves a high degree of accuracy by using a custom coding and a partial serialization of the filtering, in order to reduce the number of mapped physical resources. The design, however, is very tailored for DoG, since it exploits the separability of Gaussian kernels, and it can be hardly generalized to generic VS methods. In [12], the design does not exploit the separability of Gaussian kernels but proposes a complex arrangement of SRAMs to implement sliding window and row-shuffling operation by means of a switching network whose complexity increases with the filter dimensions. On the other hand, very few optimizations have been addressed to the arithmetic units, which are usually simplified by recurring to fixed-point (FI) codes in place of 32 bits floating-point (FP32), with impact on the accuracy of the overall VS systems. Additionally, almost all published HW solutions use surrogates of the SW conterparts, as in the case of DoG calculations that work on a reduced number of scales and octaves [13] with respect to the optimal one [5] or with Gaussian kernels having reduced standard deviations [14].

With the purpose to provide a 2D convolution-based filter that can improve VS applications in terms of area, power constrains and elaboration velocity, in this work a new design is proposed, which is capable to outperform existent FP32 implementation of 2D filters, by exploiting a new partitioning method of the operands [15], in the case that one of that assumes multiple constant values. The design works on a continuous stream of data, directly provided by the input source, and avoids the use of frame buffers by means of a careful organization of small intermediate buffers. The accuracy of the elaborated results is ensured by the use of FP32 coding, while its compactness is given by the complete absence of multiplier circuits. Although the proposed design can work with a large number of filter kernels, in this work it has been addressed to Gaussian filtering, in order to demonstrate its advantages in one of the most diffused and computational demanding application. Implementation of the 2D filter on a high-end FPGA returns a total delay path of 4.7 *ns* to produce an IEEE-754 FP32 result, starting from a window of 3x3 pixels, while std_cell implementation with TSMC CMOS 90 *nm* technology, returns 4.4 *ns*, both in slow/slow corner.

All the authors are with the Department of Industrial Engineering (D.I.In.), University of Salerno, Via Giovanni Paolo II, 132, Fisciano, Salerno, Italy. (e–mails: gdlicciardo@unisa.it, ccappetta@unisa.it, ldibenedetto@unisa.it, arubino@unisa.it, rliguori@unisa.it).

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

TABLE I
APPLICATION OF THE PROPOSED PARTITION METHOD

Input	*Partition*						
	3^0	3^1	3^2	3^3	3^4	...	λ_n
0	0	0	0	0	0	...	0
1	+1	0	0	0	0	...	0
2	-1	+1	0	0	0	...	0
3	0	+1	0	0	0	...	0
4	+1	+1	0	0	0	...	0
5	-1	-1	+1	0	0	...	0
...	...	...	...	...	...	...	...
q	C_0	C_1	C_2	C_3	C_4	...	C_n
...	...	...	...	...	...	...	...
r	+1	+1	+1	+1	+1	...	+1

II. THE UNDERLYING METHOD

The proposed partitioning method is based on the ancient mathematical problem dealing with the least number of pound weights that can be used on a scale pan to weigh any integer number of pounds from 1 to 40 inclusive [16]. Recently, this problem has been resumed and generalized when 40 has been susbtituted with a generic integer m [17]-[19]. In [17] an important proposition has been demonstrated:

Every integer weight l with $0 \le l \le m$ can be measured on a two scale balance using weights from the *multiset* $W_m := \left\{1, 3, 3^2, \ldots, 3^{n-1}, m - \left(1 + 3 + 3^2 + \ldots + 3^{n-1}\right)\right\}$.

In particular, if m=0.5(3^{n+1}−1) then W_m is the only and the smallest multiset of weights that satisfies the above Bachet problem. Considering a partition of a positive integer m, defined as an ordered sequence of positive integers that sum to m: $m = \lambda_0 + \lambda_1 + \ldots + \lambda_n$ with $\lambda_0 \le \lambda_1 \le \ldots \le \lambda_n$, this is called a *Bachet partitions* if:

1. every integer $0 \le l \le 2m$ (or $0 \le l \le m$) can be written as $l = \sum_{i=0}^{n} C_i \lambda_i$ where $C_i \in \{0,1,2\}$ (or $C_i \in \{-1,0,1\}$);
2. there not exists another partition of m satisfying 1. with fewer parts than n+1.

Such partitions are also called *minimal 2-complete partitions* [18]. The existance and the minimum number of parts composing a Bachet partition are demonstrated by the following results [18], [19]:

- *Lemma:* If $m = \lambda_0 + \lambda_1 + \lambda_2 + \ldots + \lambda_n$ is a 2-complete partition then λ_0=1 and $\lambda_i \le 1 + 2\left(\lambda_0 + \lambda_1 + \ldots + \lambda_{i-1}\right)$ for every i.
- *Corollary:* If $m = \lambda_0 + \lambda_1 + \lambda_2 + \ldots + \lambda_n$ is a 2-complete partition then $\lambda_i \le 3^i$.
- *Theorem 1:* a Bachet partition of a positive integer m has precisely $\lfloor \log_3(2m) \rfloor + 1$ parts.
- *Theorem 2:* the partition $m = \lambda_0 + \lambda_1 + \lambda_2 + \ldots + \lambda_n$ is a Bachet partition if and only if $n = \lfloor \log_3(2m) \rfloor$, λ_0=1 and $\lambda_i \le 1 + 2\left(\lambda_0 + \lambda_1 + \ldots + \lambda_{i-1}\right)$ for every i.

The uniqueness of the Bachet partition has been finally demonstrated in [20]. The principal conclusions of the above mathematical derivations that turn useful in our design, can be summarized as follows:

- Given a range of integers [0;r], it is possible to define a set of integer values, called *parts*, $S_r = \{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n\}$ of cardinality $n+1 = \lfloor \log_3(2r) \rfloor + 1$, such that all the values in the range could be obtained by a combination of λ_i.
- The parts are given by the first n powers of 3 plus $R = r - (3^0 + 3^1 + 3^2 + \ldots + 3^{n-1})$, namely, $S_r = \{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n\} := \{3^0, 3^1, 3^2, \ldots, 3^{n-1}, R\}$.
- The partition is *unique* and does not exist another partition of [0;r] satisfying the aforementioned conditions composed by *fewer parts* than n+1.

From the above results it follows that, defined the coefficients set $C := \{-1, 0, 1\}$, every term $q \in [0; r]$ can be rewritten as the superposition of the minimum number of parts:

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

Table I shows the application of the proposed partitioning method. For example: for an 8 *bits* input, the parts are $\{0, 1, 3, 9, 27, 81, 134\}$; the input 23 can be rewritten as 23=(-1)1+(-1)3+(0)9+(+1)27+(0)81+(0)134, namely the set of values from Table I will be $\{-1, -1, 0, +1, 0, 0\}$.

Eq. (1) can be employed to partition a generic 2D-convolution between a kernel F, having $K \times K$ dimensions, and an input matrix $I(x,y)$. The filtered output, $O(x_0,y_0)$, at the point (x_0,y_0), can be calculated as:

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

where $F(h,j)$ and $C_i(h,j)$ are the kernel coefficient and the sign coefficient at the generic position (h,j), respectively. Considering that typical VS applications work on integer inputs coded with a small number of bits (e.g. 8 *bits* for Luma and Chroma) and that the coefficients of the filters are *a priori* known once that the filter dimensions have been defined, the values of the n inner products between F and λ_i can be precalculated for each input value. Therefore, (2) can be simplified in a summation of precalculated coefficients, $P(h,j)$:

$$O(x_0, y_0) = \sum_{h=0}^{K-1} \sum_{j=0}^{K-1} P\left(x_0 + h - \frac{K-1}{2}, y_0 + j - \frac{K-1}{2}\right) \tag{3}$$

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

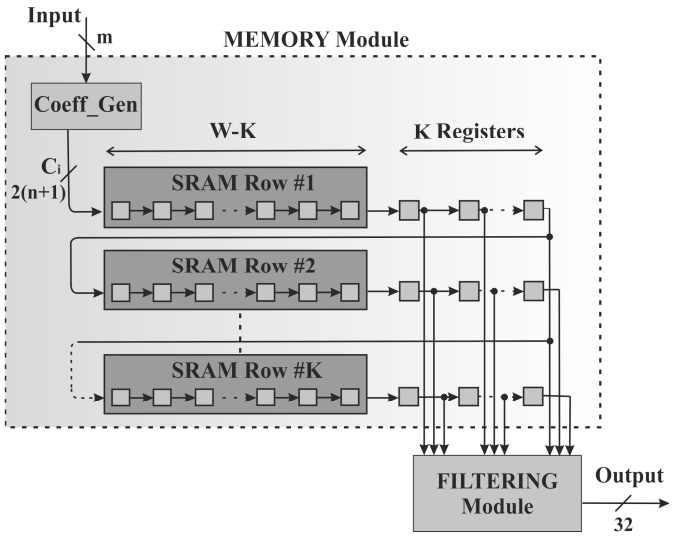

Fig. 1: Block diagram of proposed design. The Memory Module, enclosed in the dashed square, is detailed in its components.

It is worth to note that the proposed partitioning method is similar but substantially different from the Distributed Arithmetic (DA) method. DA, indeed, simplifies the calculations by recurring to a power of two partitioning, which substitutes multiplications with shifts and additions; on the contrary, the computational complexity of (3) is simplified by the smaller number of coefficients ensured by the lower number of parts of the proposed method. A radix-2 DA partitioning, for example, requires that I in (2) is decomposed as $I=\sum_{i=0}^{s-1} b_i 2^i$, where $b_i \in \{0,1\}$ represents the sign digit. Namely, (2) can be DA partitioned as:

$$O(x_0,y_0) = \sum_{h=0}^{K-1}\sum_{j=0}^{K-1}\sum_{i=0}^{s-1}\left(F(h,j)2^i\right)b_i\left(x_0+h-\frac{K-1}{2}, y_0+j-\frac{K-1}{2}\right) \quad (4)$$

Although the use of b_i in place of C_i contributes to reduce some "glue" logic to implement (4), the value of s in (4) *linearly* increases with the codelength of the input. Therefore, the number of operators to implement the inner products in (4) rapidly overcomes that of (2), where n increases with a log_3 slope. For example, in the case of 8 *bit* inputs, s=8 and n=5. Anyway, considering that the DA-related literature offers several optimized implementations, the proposed filter has been implemented with *Modified Booth* (MB) multipliers, selected as one of the best exponent of the DA-related arithmetic. Results are reported in Table III.

III. Architecture Design

The block diagram of the proposed architecture is shown in Fig. 1. It has been divided in two sequential modules, following the data flow: the *memory module* that codes the input data according to the proposed partitioning method and manages the elaboration flow, and the *filtering module* that calculates (3). Input pixels, coded as m bits unsigned integers (Uint-m), can be acquired in raster scan order directly from an image source (e.g. image sensor), without any additional caching apparatus other than that provided by the source itself. The output is an IEEE-754 compliant, FP32 filtered value, sequentially provided with a throughput coherent with the input acquisition rate. Depending on the VS algorithm used in conjunction with the proposed design, optional serdes circuitry can be added with the purpose to align the filtered pixels in a parallel fashion.

A. Memory Module

The operation principle of the *memory module* is schematized in Fig. 1. Input pixels are acquired by the *Coeff_Gen* component, essentially composed by a ROM implementing Table I, by which data are coded in a sequence of ternary sign coefficients, $C_i \in \{-1,0,1\}$. Considering that each Uint-m input must be partitioned in $n+1=\lfloor \log_3(2^{m+1}) \rfloor +1$ parts, and that 2 *bits* are needed to code each sign, the length of the resulting code is $l_C = 2\lfloor \log_3(2^{m+1}) \rfloor + 2$; namely, if m=8 *bits*, l_C=12 *bits*. In turn, the original value of the pixel is no more necessary for the subsequent calculus.

Coded data are fed in a SIPO (Serial-Input Parallel-Output) buffer, which serially stores the 1D filtered rows and outputs a $K\times K$ matrix of data to be convolved with a kernel having the same dimensions. Since input pixels are received in raster scan order, the SIPO is, in principle, folded like a stripe buffer of dimensions $K\times W$, in order to store the first K rows of the image to be processed, having width W. When K-1 rows and the first K values of the K^{th} row are serially pushed into the buffer, the rightmost K columns of the buffer can be filtered in parallel.

It is important to note that, the above organization allows that, each time a new value is pushed into the buffer, all data shifts so that those to be filtered are "naturally" aligned in the rightmost columns, without auxiliary circuitry to realize row-shuffling operations [12]. Although a straightforward implementation of the stripe buffer, by using registers, is technically possible, it is strongly deterred for the large amount of physical resources required. For example, with reference to an 8 *bits* VGA image (W = 640), the stripe of a kernel with K=25 would store 640×25=16'000 values, each coded with 12 *bits*, corresponding to 188 *kbits*. A much more suitable solution consists in using SRAM to "emulate" the SIPO behavior of the buffer. Given the availability of embedded SRAM modules, both in standard cell technology and FPLs, this solution enables the implementation of the processor in both kind of target platforms. In order to enable the writing and reading of data during the same clock cycle, each row of the stripe buffer has been implemented by a dual-port SRAM of dimensions $l_C\times(W-K)$ to store almost a complete frame row. The parallel reading of the rightmost $K\times K$ data has been implemented by completing each SRAM row with K registers, connected like in Fig. 1, in order to preserve the shift operations needed by the stream of data. It is worth to note that, due to the casual access of SRAMs, it makes no more sense to speak of the shifting operation of data, but a correct generation of addresses emulates the *shift* of the stored values and their alignment. It is worth to underline that the proposed solution is always advantageous, in terms of total memory requirement, with respect to a frame buffer-

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

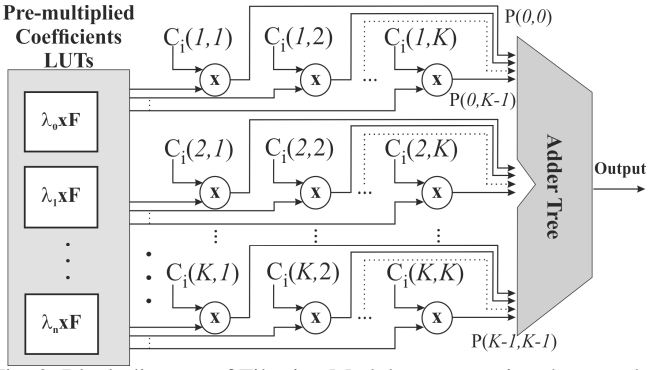

Fig. 2: Block diagram of Filtering Module, representing the way the "equivalent" multipliers are interconnected.

based implementation, since it requires a fraction of the memory of a complete frame.

B. Filtering Module

Fig. 2 shows the block diagram of the *Filtering Module*, representing the organization and interconnections of the *equivalent* $K\times K$ multipliers. The K small LUTs store the $(n+1)$ parts pre-multiplied by the K coefficients of the filter, $F(h,j)\lambda_i$ in (2), coded with length l_S, calculated in the following. In order to simplify the adder structure, the K LUTs are used to store also the 2's complement of the pre-multiplied coefficients, which are selected when $C_i=-1$, without additional overhead. Therefore, each LUT has dimensions $2(n+1)\times l_S$. The structure of a single *equivalent* multiplier is shown in Fig. 3. It has been substituted by n adders distributed along a $\lceil \log_2(n+1) \rceil$ depth tree, which calculates (3) by using the pre-multiplied coefficients, selected by a multiplexer bank, and the C_i coefficients provided by the stripe buffer.

Even if the adders should have, in principle, a FP32 architecture, a custom coding has been adopted for partial results, achieving a reduction of the adders' complexity, without altering the accuracy of the multiplication. Starting from the standard IEEE-754 coding, all the exponents of the pre-multiplied coefficients have been increased to that of the greatest one, the significands have been shifted accordingly and their length has been increased to include the shifted codes without truncations. In particular, if F^{min} and F^{max} are the minimum and maximum kernel coefficients, respectively, the codelength of the significands is increased to a number of bits:

$$l_S = \left\lceil 23 + \log_2\left(\frac{F^{\max}\lambda_n}{F^{\min}\lambda_0}\right)\right\rceil \qquad (5)$$

where 23 *bits* is the length of the standard FP32 significand. Therefore, the normalization of the mantissa after every intermediate addition is avoided and the exponent, as well as the devoted circuitry, can be omitted. A normalization stage has been introduced at the end of the overall computation to normalize the output in a standard FP32 format.

TABLE II
CUSTOM CODING APPLIED TO THE SMALLEST PRE-MULTIPLIED COEFFICIENT WITH UINT-8 INPUTS AND σ=4

Smallest Coeff.	*r*	λ_n	*FP32 coding of* 1.228x10^{-6}
1.1x10^{-3}	256	134	00110101101001001100001101010100
			Modified coding of 1.228x10^{-6}
			00000000000000000000101001001100001101010100

*from (3) the significand must be enlarged of 21bits.

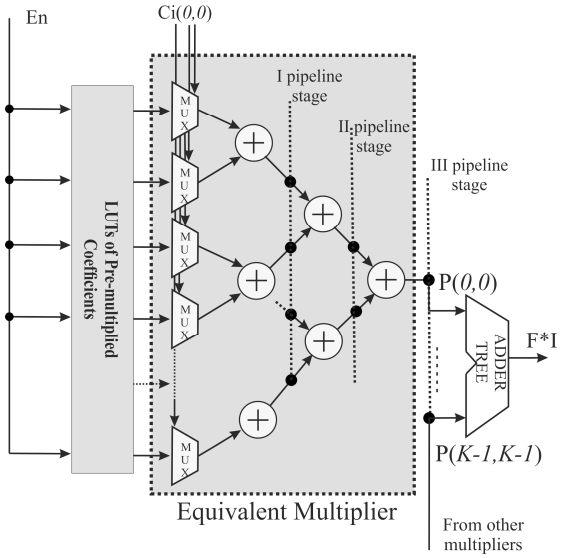

Fig. 3: Block diagram of multiplier, implemented with the proposed partitioning method.

IV. SYNTHESIS AND RESULTS

In order to contextualize the proposed design in a typical VS scenario and make the derived results comparable with the existent literature related to VS applications, the processor has been implemented with a 2D symmetric Gaussian kernel, $G(x,y,\sigma)=(2\pi)^{-1}\sigma^{-2}e^{-\frac{x^2+y^2}{4\sigma^2}}$ working with Uint-*8* inputs. A "building block" kernel with K=3 has been implemented, since it is the minimum usable dimension for VS applications. The range of input values that can be represented is r=256, therefore, the number of parts is $n+1=\lfloor \log_3(2r) \rfloor+1=6$, given by $S_r=\{1,3,9,27,81,134\}$, and l_c=12 *bits*.

Considering that:

$G^{\max}=G(0,0,\sigma)=(2\pi)^{-1}\sigma^{-2}$,

$G^{\min}=G(3,3,\sigma)=(2\pi)^{-1}\sigma^{-2}e^{-\frac{18\sigma^2}{2\sigma^2}}=(2\pi)^{-1}\sigma^{-2}e^{-9}$, $\lambda_n=134$ and $\lambda_0=1$, from (4) the length of the significands must be $l_S=\left\lceil 23+\log_2\left(\lambda_n e^9\right)\right\rceil=44\,bits$. An example of the above recoding is shown in Table II. Derived results can be easily generalized to greater dimensions by using curves in Fig. 4, by which only the dimensions of *Coeff_Gen* have been omitted since they depend only on the input coding.

The design has been targeted to a Xilinx Virtex 7 XC7V2000tflg1925-1, as part of the proFPGA DUO ASIC prototyping board [21] and to TSMC CMOS 90nm std_cells. Synthesis results have been reported in Table III and compared with implementations of the filter using conventional FP32 and MB multipliers, all targeted to the same FPGA and std_cells. In order to present a fair comparison and as much reproducible and comparable results

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

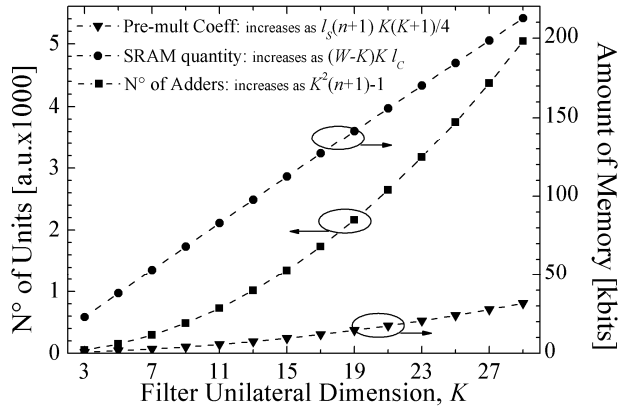

Fig. 4: Required resources of the 2D convolution-based filter as a function of its dimensions, when m=8 bits and W=640 pixels.

as possible, IPs provided by Xilinx for the adders and the multipliers, both configured with a 3-stage pipeline, have been used as building components of the proposed design. For the same reason, we did not impose aggressive constrains: the most relevant, in the case of FPGA, concerns the exclusion of embedded DSPs and in the forced use of Block RAMs. In addition, the *flatten hierarchy* option has been disabled and a general synthesis strategy toward speed has been selected. No particular constrains have been set for std_cell implementation. The post place&route data in Table III shows that the FPGA is the most advantageous platform to implement the proposed multiplier. Indeed, considering that at regime, one pixel is filtered per clock cycle, the FPGA implementation exhibits a speed-up of 371% with respect to a conventional multiplier, whereas the worst path delay reduces from 17.432 *ns* to 4.700 *ns* in the slow/slow corner, and a speed-up of 215% with respect to an MB multiplier, where the delay increases to 8.875 *ns,* also for the absence of DSPs. The mapped physical resources are approximately lower of 38.6% than in the conventional case and 44.8% than in the MB-based filter, while the normalized dissipated power is approximately 25% and 45% lower than the conventional and the MB one, respectively. The extremely positive results of FPGA implementation are justified by its very short critical path that, in the worst case, involves three pipelined adders and two parallel memory access. Memories have been implemented by means of Block-RAMs that, in the targeted Virtex 7 with speed grade -1, present an access time of 2.18 *ns.* The routing delay has been reduced by a very low congestioned floorplanning, allowed by the reduced number of CLBs mapped for the logic. This is also confirmed by the results of the std_cell implementation, where it is observed a reduction of about 24% in area and a speed-up of 94.75% with respect to a conventional multiplier. The absence of very optimized

TABLE IV
COMPARISON OF THE PROPOSED DESIGN WITH THE RELATED LITERATURE

	Std_cells		***FPGA***	
	*Prop**	*Huang [12]*	*Prop***.*	*Cabello [27]*
Technology	CMOS 180nm	CMOS 180nm	Spartan 6	Spartan 6
Output Resolution	FP24	Fixed 24 bits	FP16	FP16
LUTs	--	N.A.	2395	5052
Mem. [kbits]	255	224	4	1 BRAM
Max freq.[MHz]*	126	100**	145	100

*scaled to 3 octaves, 6 scales. 3 stage pipeline. **extracted by the overall system velocity ***3x3 kernel

TABLE III
SYNTHESIS RESULTS OF THE PROPOSED FILTER COMPARED WITH THOSE BASED ON CONVENTIONAL AND MODIFIED BOOTH MULTIPLIERS

	FPGA			***Std_cells***		
	Prop.	*Conv. FP32*	*MB*	*Prop.*	*Conv. FP32*	*MB*
Technology	XC7V*	XC7V	XC7V	90nm	90nm	90nm
LUTs/ Area[mm²]	4750	7732	8606	0.294	0.387	0.51
Mem. [byte]	582	--	528	582	--	--
*Delay**[ns]*	4.700	17.432	8.785	4.426	8.717	4.483
*Power*** [W]*	0.684	0.907	1.226	0.014	0.0136	0.019

*Virtex 7 **1 pixel filtered per clock cycle ***Normalized at 100MHz

memory modules and devoted interconnections reduces the speed-up of the proposed design, which is quite the same than that of the MB-based design. The power dissipation is 2.84% higher than that of the conventional case, mainly due to the consumption of the memories. In obtaining the data in Table III, it has been considered that all the LUTs must be read from all the multipliers on the same clock edge. Although this can be easily implemented in FPGA, ASICs require a custom implementation of very small ROMs, developed in a way similar to the one presented in [22]. However, the amount of required memory does not represent an actual problem in real multimedia applications, whereas the memory requirement is in the order of *Mbits* because of frame buffering [23], large tables [24], [25] and partial data storage [26], which makes negligible the additional area required. A direct comparison of the results in Table III with the existent literature is very hard, because the search for the reduction of HW complexity has lead almost all authors to implement HW designs with fixed-point arithmetic or floating-point with reduced accuracy. Therefore, a fair comparison with the existent literature has been possible only by scaling-down the proposed design with respect to the results in Table III. Comparison with the 2D Gaussian filter for SIFT in [12] has been obtained by a scaling-down to FP24 and a synthesis with TSMC 180nm std_cell libraries. In turn, comparison with the design in [27] has been carried out by using the same Xilinx Spartan 6 FPGA [28] used in the Nexys 3 board and scaling-down the accuracy to FP16. Results considering the available data are reported in Table IV. The greater quantity of memory with respect to [12] is justified by the use of a FP coding and is by far compensated by the lower number of arithmetic circuits and the more accurate coding with a reduced amount of additional physical resources. For example, the adoption of a FP32 coding would require only 21 *kbits* of additional memory. In all the comparisons, the proposed design exhibits a much higher elaboration speed, although the absence of a very optimized memory path of the lower-end FPGA, reduces the achievable speed-up.

As a final observation, it is important to underline that the

TABLE V
COMPARISON OF THE PROPOSED DESIGN WITH A GAUSSIAN AND A WEIGHTED AVERAGE KERNEL

	FPGA		***Std_cells***	
	Gaussian	*W. Aver*	*Gaussian*	*W. Aver*
Technology	XC7V*	XC7V	90nm	90nm
LUTs/ Area[mm²]	4750	4744	0.294	0.293
Mem. [byte]	582	483	582	483
*Delay**[ns]*	4.700	4.877	4.426	4.669
*Power*** [W]*	0.684	0.678	0.014	0.010

*Virtex 7 **1 pixel filtered per clock cycle ***Normalized at 100MHz

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2016.2603068, IEEE Transactions on Circuits and Systems for Video Technology

use of a Gaussian filter is the most demanding in term of resources instantiated by the proposed design. Indeed, the dimensions of the tables for pre-multiplied coefficients are strictly related to the kernel dimensions that, in turn, must be significantly larger than the standard deviation of the kernel. Therefore, the use of a different kernel generally causes a significant memory reduction. In turn, the arithmetic complexity of the filtering remains unchanged, since there are no multiplications and the number of additions only depends on the ranges of input values and of kernel coefficients. What said is confirmed from the results in Table V where the proposed Gaussian-based implementation of Table III has been compared with a 2D weighted average filter, having FP32 coded real weights, taken as representative of large number of VS filters [10]. The only notable difference is in the reduction of about 17% of the memory required for coefficients. Naturally, this percentage increases with the kernel dimensions.

V. Conclusion

In this paper, a new HW architecture has been presented for 2D convolution-based filtering of images and video-frames. It is particularly useful for VS applications, where performances strongly contrast with the number of arithmetic operators and required memory. Both the memory and the arithmetic apparatus have been design in order to improve the throughput and the amount of mapped resources. The memory compartment has been designed to elaborate images in raster scan order, without internal or external frame buffers. In turn, a new partitioning method has been used to improve the arithmetic compartment that substitutes multipliers with simplified adders and ROMs for storing pre-multiplied coefficients. The proposed solution obtains state-of-the-art performances in both std_cells and FPGA target platforms. In addition, power dissipation keeps to values that justify the employment of the processor for handheld, portable devices.

References

[1] J. Luo and G. Oubong, "A comparison of SIFT, PCA-SIFT and SURF", *International Journal of Image Processing*, Vol. 3, No. 4, pp. 143–152, Aug. 2009.

[2] S. L. Chen, "VLSI implementation of an adaptive edge - enhanced image scalar for real - time multimedia applications", in *IEEE Trans. on Circuits and Systems for Video Technology,* Vol. 23, No. 9, pp.1510–1522, Sep. 2013.

[3] M. Basu, "Gaussian-Based Edge-Detection Methods—A Survey", in *IEEE Trans. on Systems, Man and Cybernetics - Part C: Applications and Reviews,* Vol. 32, No. 3, pp.234–240, Aug. 2002.

[4] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features", *in Computer Vision, 1999. The Proc. of the Seventh IEEE International Conf. on,* Kerkyra, Vol. 2, pp.1150–1157, Sep. 1999.

[5] D. G. Lowe, "Distinctive image features from scale-invariant key points", in *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Jan. 2004.

[6] K. Mizuno et al., "A low power real-time SIFT descriptor generation engine for full hdtv video recognition", in *IEICE Trans. Electron*, Vol. E94-C, No. 4 Apr. 2011.

[7] M. Grabner, H. Grabner, and H. Bischof, "*Fast approximated SIFT*", in Asian Conf. on Computer Vision, Hyderabad, India, 2006.

[8] J. Jiang, X. Li, and G. Zhang, "SIFT Hardware Implementation for Real-Time Image Feature Extraction", in *IEEE Trans. on Circuit and Systems for Video Technology,* Vol. 24, No. 7, pp. 1209–1220, Jul. 2014.

[9] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk and B. Girod, "CHoG: Compressed Histogram of Gradients a Low Bit-Rate Feature Descriptor", in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on,* Miami, FL, pp. 2504–2511, Jun. 2009.

[10] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features", in *Proc. 9th Eur. Conf. Comput. Vis*., 2006, pp. 404–417.

[11] G.D. Licciardo, T. Boesch, D. Pau and L. Di Benedetto, "Frame Buffer-less Stream Processor for Accurate Real-Time Interest Point Detection", in *Integration, the VLSI Journal*, Vol.54, pp. 10–23, Jun. 2016.

[12] F. C. Huang, S.Y. Huang, J. W. Ker and Y. C. Chen, "High performance SIFT hardware accelerator for real - time image feature extraction", in *IEEE Trans. on Circuit and Systems for Video Technology,* Vol. 22, No. 3, pp. 340–351, Mar. 2012.

[13] N. P. Borg, C. J. Debono, D. Zammit-Mangion, "A Single Octave SIFT Algorithm for Image Feature Extraction in Resource Limited Hardware Systems", in *Visual Communications and Image Processing Conf.,2014 IEEE*, Valletta, pp. 213–216, Dec. 2014.

[14] E. S. Kim and H. J. Lee, "A novel hardware design for SIFT generation with reduced memory requirement," *J. Semicond. Technol. Sci.,* Vol. 13, No. 2, pp. 157–169, Apr. 2013.

[15] G. D. Licciardo, C. Cappetta, L. Di Benedetto, M. Vigliar, "Weighted Partitioning for Fast Multilpier-less Multiple Constant Convolution Circuit", *IEEE Trans. on Circuits and Systems II: Express Briefs,* in-press, doi: 10.1109/TCSII.2016.2546899.

[16] E. O'Shea, "Bachet's problem: as few weights to weigh them all", *arXiv: 1010.5486*, pp. 1 - 15, Oct. 2008.

[17] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers (sixth edition)*, Oxford University Press, 2008.

[18] S. K. Park, "The r-complete partitions", *Discrete Mathematics*, No. 183, pp. 293–297, 1998.

[19] Ø. J. Rødseth, "Enumeration of M-partitions", *Discrete Mathematics*, No. 306, pp. 694–698, 2006.

[20] P.A. MacMahon, "Combinatory Analysis". (vols. 1 & 2)(III Ed.), *AMS Chelsea Publishing*, 1984.

[21] Virtex - 7 Family, DS183 (v1.23), Xilinx, San Jose, CA, USA, Jun. 23, 2015.

[22] B. C. Paul, S. Fujita and M. Okajima, "ROM - Based Logic (RBL) design: a low - power 16 bit multiplier", in *IEEE Journal of Solid - State Circuits*, Vol. 44, No. 11, pp.2935–2942, Nov. 2009.

[23] W. M. Chao and L. G. Chen, "Pyramid architecture for 3840x2160 Quad Full High Definition 30 frames/s video acquisition", *IEEE Trans. on Circuits and Systems for Video Technology,* Vol. 20, No. 11, pp. 1499–1508, Nov. 2010.

[24] G. D. Licciardo and M. Costagliola, "An H.264 Encoder for Real Time Video Processing Designed for SPEAr Customizable System-on-Chip Family," Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on, Dubai, 2007, pp. 824-827. doi: 10.1109/ICSPC.2007.4728446.

[25] G.D. Licciardo, L.F. Albanese, "Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms", *IET Image Processing*, vol.6, no.4, pp. 301-308, June 2012.

[26] G. D. Licciardo, A. D'Arienzo and A. Rubino, "Stream processor fo real - time inverse Tone Mapping of Full - HD images", in *IEEE Trans. on VLSI Systems,* Vol.23, No. 11, pp. 2531–2539, Nov. 2015.

[27] F. Cabello, J. Leon, Y. Iano and R. Arthur, "Implementation of a Fixed-Point 2D Gaussian Filter for Image Processing Based on FPGA", in *Signal Processing: Algorithms, Architectures, Arrangements and Applications (SPA)*, pp. 28–33, Sep. 2015, Poznan.

[28] Spartan - 6 Family, DS160 (v2.0), Xilinx, San Jose, CA, USA, Oct. 25, 2011.

1051-8215 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.