

# Weighted Partitioning for Fast Multiplier-less Multiple Constant Convolution Circuit

Gian Domenico Licciardo, *Member, IEEE*, Carmine Cappetta, *Student Member, IEEE*,  
Luigi Di Benedetto, *Member, IEEE* and Mario Vigliar

**Abstract**— A new radix-3 partitioning method of natural numbers, derived by the weight partition theory, is employed to build a multiplier-less circuit well-suited for multimedia filtering applications. The partitioning method allows to conveniently pre-multiply 32 bit floating-point (FP32) filter coefficients with the smallest set of parts composing an unsigned integer input. In this way, similarly to the Distributed Arithmetic, shifters and recoding circuitry, typical of other well-known multiplier circuits, are completely substituted with simplified floating-point adders. Compared to the existent literature, targeted to both FPGA and `std_cell` technology, the proposed solution achieves state-of-the-art performances in terms of elaboration velocity, achieving a critical path delay of about 2 ns both on a Xilinx Virtex 7 and with CMOS 90nm `std_cells`.

**Index Terms**— Multiplier, Convolution, Gaussian filter, Distributed arithmetic.

## I. INTRODUCTION

Recent advancements in the elaboration of high-quality media contents have promoted an intense research activity for the improvement of filtering operators, whose hardware (HW) complexity is a major concern in applications aimed to pure speed, as for example, image and video elaboration [1], [2]. Such complexity, indeed, usually relapses in the allocation of a large number of arithmetic operators and a consequent slackening of the overall circuit. The recent literature shows that the above issue is usually managed either by recurring to the full/partial serialization of the filters [3],[4] and folding techniques [5], or by intervening on the intrinsic complexity of fused Multiply-Adders and Multiply-ACcumulators (MAC). Since the former way usually causes a significant reduction of the filter performances [6], the latter approach remains the most accurate way of achieving a good Power, Performances, Area (PPA) trade-off. In this case, the complete removal of the multiplier circuitry is by far the preferred choice of several authors [6], [7], which recur to fast adders and shifters in place of multipliers, according to the coding of the operands, Canonical Signed Digit (CSD) and Modified Booth (MB), primarily [8],[9]. The simplification of filtering circuits becomes particularly effective when one of the operand can be reduced to a bounded set of pre-calculated values, as in the case of pre-defined filter kernels. In such

G.D. Licciardo, C. Cappetta and L. Di Benedetto are with the Department of Industrial Engineering (D.I.In.), University of Salerno, Via Giovanni Paolo II, 132, Fisciano, Salerno, Italy. (e-mail: gdlicciardo@unisa.it, cappetta.carmine@gmail.com, ldibenedetto@unisa.it).

M. Vigliar is with Spark srl, via Antonio Gramsci 86/A Reggio Emilia, Italy, (e-mail: mario.vigliar@spark-security.com).

cases, the Distributed Arithmetic (DA) method [10] can be successfully applied in order to partition multiplications in simpler shifts and additions. By using memories to store pre-calculated partial sums, whose number can be reduced by the help of Multiple Constants Multiplications (MCM) techniques [11], [12], DA can be, in principle, advantageously used in place of MB and CSD [13]. However, actual performances of DA result from a careful trade-off between its “natural” bit-serial operation and the parallelism by which the partial sums are calculated, which can lead to the excessive increment of mapped physical resources [10].

In this work, a new partitioning scheme is proposed, based on radix-3 terms (called *parts*), derived by the solution of the weight problem [14], with the purpose to improve the performances of DA, as well as the performances of general purpose multipliers in the specific contexts of MCM. Although it is based on the same operation principle of DA, we will demonstrate that the proposed method is always advantageous in terms of mapped physical resources and elaboration speed, for multiplying 32 bit floating-point (FP32) filter coefficients with integer inputs. In this way, shifters and recoding circuitry, typical of other well-known multiplier scheme, can be completely substituted by floating-point adders, whose HW complexity can be simplified to that of fixed point (FI) adders, without undermining the accuracy. The derived implementation is adequate for multiple-constant filtering applications, working with FP pre-calculated kernel coefficients and input quantities included in a range of integer values compatible with multimedia elaboration. Implementation of the proposed *multiplier* on a high-end FPGA returns a total delay path of 2.456 ns to produce an IEEE-754 FP32 [15] result, starting from a 8 bit unsigned integer input, while `std_cell` implementation with TSMC CMOS 90 nm technology, returns 2.61 ns, both in slow/slow corner. These delays are significantly lower than those achievable by conventional FP32 multiplier implemented with the same platform, and working on the same dataset, while they are comparable with `std_cells` implementations in 65 and 45 nm CMOS technology [16], [17].

## II. THE UNDERLYING PARTITIONING METHOD

The problem of establishing the least number of integers and their values such that all the numbers in a limited range can be expressed as a combination of them has been faced since XVII century, because it finds a large spectrum of applications, including the simplification of the circuitry devoted to filtering apparatus [18]. About this argument,

TABLE I  
APPLICATIONS OF THE PROPOSED PARTITION METHOD

Input	Partition						
	$3^0$	$3^1$	$3^2$	$3^3$	$3^4$	...	$\lambda_n$
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

several particular demonstrations have been published [19]-[21]. A general resume has been published in [14], where it has been definitively demonstrated that, having defined the set  $W_r := \{3^0, 3^1, 3^2, \dots, 3^{n-1}, R\}$  with  $R = r - (3^0 + 3^1 + 3^2 + \dots + 3^{n-1})$ , every integer in the range  $[0; r]$  can be obtained by a superposition of the terms in  $W_r$  multiplied for a coefficient  $C_i \in \{-1, 0, 1\}$ .

More in general, defined as *partition* of a positive integer  $r$  the ordered sequence of positive integers that sum to  $r = \lambda_0 + \lambda_1 + \dots + \lambda_n$  with  $\lambda_0 < \lambda_1 < \dots < \lambda_n$  and as *parts* of the partition the set  $\{\lambda_0, \dots, \lambda_{n-1}, \lambda_n\}$ , it can be demonstrated that:

- every integer  $0 \leq q \leq r$  can be written as:

$$q = \sum_{i=0}^n C_i \lambda_i \quad (1)$$

- there *does not exist* another partition of  $r$  satisfying 1 with *fewer parts* than  $n+1$ .

An important corollary of the above properties demonstrates that every partition of  $r$  is composed by exactly  $n+1 = \lfloor \log_3(2r) \rfloor + 1$  parts. Table I shows the application of the 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  $\{-1, -1, 0, +1, 0, 0\}$ .

The above results can be applied to MAC operations between a generic vector of coefficients  $A_k$  and a vector of inputs  $x_k$ :

$$y = \sum_{k=0}^{K-1} A_k x_k \quad (2)$$

by using (1) in (2), it results:

$$y = \sum_{k=0}^{K-1} \sum_{i=0}^n A_k C_{ki} \lambda_i \quad (3)$$

Eq. (3) can be rewritten in terms of the parts in  $W_r$ :

$$y = \sum_{k=0}^{K-1} \left[ \sum_{i=0}^{n-1} A_k C_{ki} 3^i + A_k C_{kn} R \right] \quad (4)$$

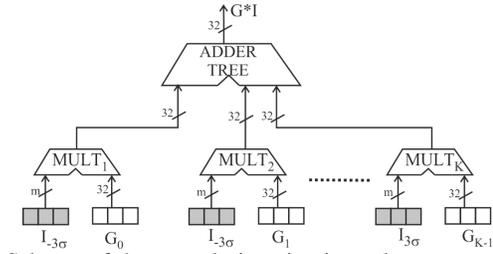


Fig. 1: Scheme of the convolution circuit used as case study of the proposed decomposition method.

Considering that  $n$  is small in many cases of interest, and that the above partition is a linear superposition of parts, the previous results suggest that the multiplications involved in a particular filter can be reorganized as the partition of pre-multiplied terms, obtained by calculating *a priori* the product between the parts and  $A_k$ . This partitioning is very different

from DA that requires that  $x_k$  is decomposed as  $x_k = \sum_{i=0}^{s-1} b_{ki} 2^i$ ,

where  $b_{ki} \in \{0, 1\}$  represents the sign digit. Namely, (2) can be DA partitioned as:

$$y = \sum_{k=0}^{K-1} \sum_{i=0}^{s-1} A_k b_{ki} 2^i \quad (5)$$

Although the use of  $b_{ki}$  in place of  $C_{ki}$  contributes to reduce some “glue” logic to implement (5), actual values of  $s$  in (5) are significantly higher than  $n$  in (4). Therefore, the number of operators to implement the inner products in (4) can be strongly reduced.

### III. ARCHITECTURE DESIGN

The proposed method has been used for implementing the scheme in Fig. 1 that calculates the convolution  $G*I$  between the FP32 kernel vector,  $G$ , and the input vector  $I$  of unsigned integer coded with  $m$  bit (Uint- $m$ ). The proposed method is used to improve the structure of the  $k$  MACs that calculate the product between the generic coefficient of the kernel and the

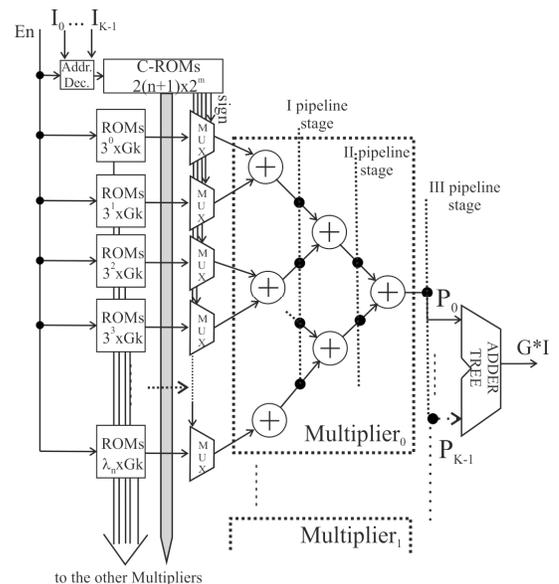


Fig. 2: Scheme of the proposed multiplier, deployed in a Gaussian convolution.

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

Smallest Coeff.	$r$	$\lambda_n$	FP32 coding of $1.1 \times 10^{-3}$
$1.1 \times 10^{-3}$	256	134	0 01110101 0010000010110111100000
			Modified coding of $1.1 \times 10^{-3}$
			000000000000100100000010110111100000

\*from (7) the significand must be enlarged of 14bits.

vector of input values. The resulting architecture is schematized in Fig. 2. Once  $G$  has been defined, each element of  $I$  can be decomposed in  $n+1$  parts, according to Table I, and stored in an equal number of dual-port Read-Only Memories (ROMs), after that they have been pre-multiplied by the elements of  $G$ . A further  $(n+1)2^{m+1}$  bit ROM stores the  $C_i$  coefficients (2 bits each one) used to select the sign of the operands. The multipliers are substituted by the  $n$  adders in the dashed box of Fig. 2, distributed along a  $\lceil \log_2(n+1) \rceil$  depth tree. In principle, the adders should have a FP architecture, but it is possible to adopt a custom coding for partial results, in order to reduce their complexity, without altering the accuracy of the multiplication. Starting from the standard IEEE754 FP32 coding [15], all the exponents of the pre-multiplied coefficients have been increased to that of the greatest one, while the number of bits of the significands have also been increased accordingly, in order to include the shifted codes and avoid truncations.

Although the proposed method can be employed in conjunction with several kind of kernels, as a case study, it has been used to implement the Gaussian filter with kernel

$G(x, \sigma) = Ae^{-\frac{x^2}{2\sigma^2}}$  for its large diffusion in image and video elaboration flow (e.g. visual search, blurring, segmentation, and so on), where they usually work with Uint-8 input (e.g. Luma or Chroma image pixel). Thanks to its separability property, for which a two-dimensional (2D) filter can be separated in two consecutive 1D products [3], [18], the filter implementation in the space/time domain is typically preferred to the frequency conversion. Results from section II allow partitioning  $I$  by using the first  $n+1 = \lceil \log_3 2(2^8 - 1) \rceil + 1 = 6$  parts from Table I, and write the convolution as:

$$\begin{aligned}
 G(x, \sigma) * I(x) &= \sum_{j=0}^{K-1} G_j I_{j - \frac{K-1}{2}} = \sum_{j=0}^{K-1} G_j \sum_{i=0}^n C_i \lambda_i = \\
 &= \sum_{j=0}^{K-1} \sum_{i=0}^n (G_j \lambda_i) C_i = \sum_{j=0}^{K-1} P_j
 \end{aligned} \quad (6)$$

In particular, all the  $n$  inner products,  $G_j \lambda_i$ , are pre-

TABLE III  
REQUIRED MEMORY AS A FUNCTION OF THE INPUT RANGE

Input length [bit]	$C_i + \lambda_n(C_i)$ [kbits]	
	Proposed	radix-2 DA
8	5.958 (3.072)	5.896 (2.048)
9	10.535 (7.168)	8.937 (4.608)
10	17.703 (14.336)	15.050 (10.240)
...	...	...
$m$	$\lambda_n = (n+1) \times [(3\sigma+1)l]$ $C_i = (n+1) \times 2^{m+1}$	$\lambda_n = m \times [(3\sigma+1)l]$ $C_i = m \times 2^m$

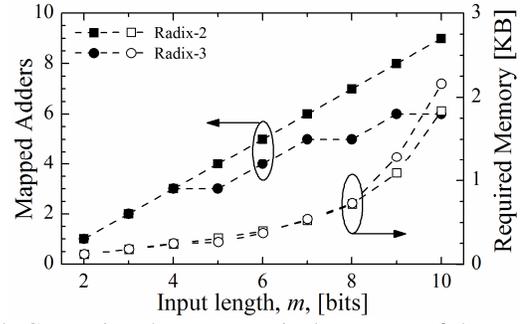


Fig. 3: Comparison between required resources of the proposed partitioning method and the radix-2 DA, with  $l=37$  and  $(3\sigma+1)=13$ .

computed for each value of  $I$  in the range  $[0;255]$ , for every kernel coefficient, since  $G_j$  remains constant once  $\sigma$  and  $K$  have been defined. Given that the kernel length can be imposed  $K=6\sigma+1$  points with good accuracy [3], and that for the Gaussian symmetry it is possible to store only  $3\sigma+1$  values, (6) can be implemented by the scheme in Fig. 2. The input  $I$  is used to access the ROMs that, in principle, are sharable by all the multipliers. Outputs from the  $C$ -ROMs provide the signals to select the inputs to the first adder's row of the multipliers. The remaining ROMs, having depth  $(3\sigma+1)$ , provide the  $G_j \lambda_i$  coefficients that must be added toward the final result. With the purpose to eliminate the handling of the exponent, considering that the greatest pre-multiplied coefficient is  $G_0 \lambda_n = \frac{\lambda_n}{\sqrt{2\pi\sigma}}$  and the smallest one

is  $G_{K-1} \lambda_0 = \frac{\lambda_0}{\sqrt{2\pi\sigma}} e^{-\frac{9\sigma^2}{2\sigma^2}} = \frac{e^{-4.5}}{\sqrt{2\pi\sigma}}$ , the codelength of significands is increased of  $\lceil \log_2 \left( \frac{G_0 \lambda_n}{G_{K-1} \lambda_0} \right) \rceil = \lceil \log_2 (\lambda_n e^{4.5}) \rceil = \lceil \log_2 (\lambda_n) + 6.5 \rceil$  bits.

Therefore, considering that  $\lambda_n \equiv R = 255 - \sum_{i=0}^4 3^i = 134$ , the codelength of the pre-multiplied significands become:

$$l = \lceil 23 + \log_2 (\lambda_n) + 6.5 \rceil = \lceil \log_2 (134) + 29.5 \rceil = 37 \text{bits} \quad (7)$$

Furthermore, in order to reduce the impact of this enlargement on the sizes of the ROMs, the exponent has been omitted from the partial codes and then re-introduced in the final result to normalize the data to the standard FP32 format. It is worth to note that the FP coding is necessary for applications requiring very high dynamic range [16], as in the case of inverse-Tone Mapping [3], where ranges higher than 70 are highly recommended. An example of the employed coding is shown in Table II, applied to the smallest coefficient with Uint-8 inputs and  $\sigma=4$ .

In Table III, the memory required by the proposed solution, detailed for  $C_i$  and  $\lambda_n$ , is compared with the corresponding quantity required by a radix-2 DA. In both cases, the mapped resources refer to full-parallel architectures with codelength  $l=37$  and  $(3\sigma+1)=13$ . A graphical representation of the

TABLE IV  
SYNTHESIS OF THE PROPOSED MULTIPLIER IN COMPARISON WITH RECENT FPGA AND STD. CELLS ORIENTED DESIGNS

	FPGA				Std cell		
	Proposed*	Conv. FP32*	32 bits MB	Arish [23]	Proposed*	Basiri [16]	Själänder** [17]
Target platform	Virtex 7	Virtex 7	Virtex 7	Virtex 4	CMOS 90nm	CMOS 45nm	CMOS 65nm
LUTs/Area [ $\mu\text{m}^2$ ]	475	612	1634	1545	21269*** (5961/15308)	64682	52000
Memory [byte]	904	52	528	--	--	--	--
Path Delay[ns]	2.456	8.235	7.882	4.77	2.61	2.18	2.5
Power**** [W]	0.909	1.113	1.139	N.A.	$2.41 \cdot 10^{-3}$	$0.366 \cdot 10^{-3}$	$9.36 \cdot 10^{-3}$
Area-Delay-Power [ $\mu\text{m}^2 \cdot \text{ns} \cdot \text{mW}$ ]	--	--	--	--	$3.75 \cdot 10^4$	$5.16 \cdot 10^4$	$1.21 \cdot 10^6$

\*Three stage pipeline \*\*Modified-Booth multiplier \*\*\*in parenthesis logic/memories \*\*\*\*Normalized at 100MHz

comparison is shown in Fig. 3, in which also the required number of additions is shown as a function of the input length,  $m$ . For  $m > 4$  the proposed solution is always advantageous in terms of required additions, whereas the required memory becomes significantly higher for  $m > 9$ , e.g. 15% for  $m = 10$ . In these cases, indeed, the lower number of parts of the proposed method is compensated by the higher number of bits for  $C_i$ . The advantages of the proposed solution become relevant when a large number of multipliers is required. In the implementation reported in the next section, with  $m = 8$  and  $\sigma = 4$ , 25 MACs are required for a full-parallel circuit; thus, using the radix-3 proposed method, it is possible to save 50 adders with respect to the radix-2 DA. Considering also that the memory is sharable between all the MACs, for implementing filters with typical kernel dimensions, the proposed solution proves to be advantageous when compared to a conventional radix-2 DA.

#### IV. SYNTHESIS AND RESULTS

In order to give a straightforward estimation of the advantage deriving by the adoption of the proposed method, a stand-alone “equivalent multiplier” has been synthesized, composed by all the memories and the adders schematized in the dashed box of Fig. 2. It has been targeted to a Xilinx Virtex 7 XC7V2000tflg1925-1, as part of the proFPGA DUO ASIC prototyping board [22] and to TSMC CMOS 90nm std\_cells. Synthesis results have been reported in Table IV and compared with a conventional FP32 multiplier and a 32 bits Modified Booth, targeted to the same Field Programmable Gate Array (FPGA). A fair comparison has been achieved by using the IPs provided by Xilinx for the adders and the multiplier, all configured with a 3-stages pipeline. Carry-Save Adders (CSA) have been used for std\_cells implementation of the proposed structure, while no conventional multiplier has been compared for the absence of an optimized multiplier in the same std\_cell technology. The work in [23], for FPGA, and [16], [17] for std\_cells have been used as comparative terms. Although they propose slightly different architectures,

TABLE V  
SYNTHESIS RESULTS OF THE 1D GAUSSIAN CONVOLUTION CIRCUIT

	FPGA		Std_cells	
	Proposed	Conv. FP32	Proposed	Conv. FP32
Platform	Virtex 7	Virtex 7	90nm	90nm
LUTs/Area	8654	15512	275339	342127
Mem. [byte]	904	--	904	--
Delay[ns]	2.981	16.986	3.99	4.531
Power* [W]	2.317	4.495	$28.011 \cdot 10^{-3}$	$16.907 \cdot 10^{-3}$

\*Normalized at 100MHz

for the base of our knowledge, these are the most recent designs in the literature that compare with the proposed one for the similarity of the purpose. It is worth to note that the actual memory mapped in our implementation has been increased from the minimum required of 763 bytes to 904 bytes, since all the pre-multiplied coefficients have been stored together with their 2's complement negative counterparts. Therefore, both the positive and the negative pre-multiplied coefficients are available at the same time for the additions and sign conversions are avoided when  $C_i = -1$ .

As expected, Table IV shows that the FPGA is the most advantageous platform to implement the proposed multiplier, thanks to the availability of *hard macros* to implement ROMs. Indeed, the FPGA implementation exhibits a speed-up of 335% with respect to a conventional multiplier, whereas the worst path delay reduces from 8.235 to 2.456 ns in the slow/slow corner. A great advantage is observed also with respect to the MB multiplier that have a path delay of 7.882 ns. It is worth to note that the ROM access does not introduce a critical delay, since it exhibits a latency of 2.1 ns, which is significantly lower than the previous value. The mapped physical resources are approximately lower of 30% than those in [23], while the delay is about one half, although this value is not representative because of the technology differences between the two target platforms. The normalized dissipated power is the 81.7% of that of the conventional multiplier.

Std\_cell implementation exhibits good results in comparison with the MB-based multiplier in [17] and the Braun fused multiply accumulator in [16], both implementing a two-stage pipeline. It is worth to observe that, although that in [16] is not a pure multiplier, it only differs of a 24 bits Carry Look-ahead adder. Although the solutions in [16] and [17] are implemented with shrunk 45nm and 65nm technology, the delay times are only 430 ps and 110 ps higher than that proposed respectively, whereas the occupied area is about 244% and 304% times the proposed one, respectively. Considering that, for the absence of devoted ROMs of adequate dimensions, all the memories have been implemented by LUTs, this is a good result that contributes to an Area-Power-Delay product of  $3.75 \times 10^4 \mu\text{m}^2 \cdot \text{ns} \cdot \text{mW}$ , which is always better than the cited solutions.

In Table V a complete circuit for 1D Gaussian convolution, composed by 25 multipliers and an output adder-tree, connected as in Fig 2, is compared with a conventional FP32 multiplier-based solution targeted to the same FPGA and

std\_cells. In this case, the FPGA speed-up to 570% is obtained, whereas the path delay reduces from 16.986 to 2.981 ns and the total amount of LUT in the design is the 44.21% less of the one obtained using conventional multipliers. Also the overall power dissipation almost halves. In std\_cells it is possible to observe a reduction of about 19.52% in area and a speed-up of about 11.93%. In this case, the power dissipation is 39.64% more than in the conventional case, mainly due to power dissipated by the memories. In obtaining the data in Table V, it has been considered that all the ROMs must be read from all the multipliers on the same clock edge. Although this can be easily implemented in FPGA, Application Specific Integrated Circuits (ASICs) require a custom implementation of very small ROMs, similarly to [6] for ROM Based Logic (RBL) multipliers. However, the amount of memory in Table IV and V does not represent an actual problem in real multimedia applications, whereas the memory requirement is in the order of *Mbits* because of frame buffering [24] or partial data storage [3],[4], which makes, *de facto*, negligible the additional area required by the multiplier's ROMs. Furthermore, considering the large amount of partial additions, the proposed architecture of the multiplier could obtain an area reduction, by the application of MCM techniques [25], which have been demonstrated effective in reducing area when a large number of redundant terms must be calculated. Nevertheless, due to the presence of fast Look-Up Tables (LUTs) in contrast with heavy carry logic, such as in conventional FP32 multipliers, the proposed system should better fit the actual and forthcoming CMOS technologies with smaller gate sizes.

## V. CONCLUSION

In this paper an efficient term - partitioning method has been shown, which allows implementing the circuitry for convolution operators, typically employed in filters, without multipliers, encoders and auxiliary circuitry. These are completely substituted by simplified adders and ROMs for storing pre-multiplied coefficients. The proposed solution obtains state-of-the-art performances. The solution is well suited for the application of multi-constant multiplication techniques, in order to further simplify the circuit topology.

## REFERENCES

- [1] 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.
- [2] 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.
- [3] G. D. Licciardo, A. D'Arienzo and A. Rubino, "Stream processor for real - time inverse Tone Mapping of Full - HD images", in *IEEE Trans. on VLSI Systems*, Vol.23, No. 11, pp. 2531 - 2539, Nov. 2015.
- [4] M. Vigliar and G. D. Licciardo, "Hardware Coprocessor for Stripe-based Interest Point Detection," U.S. Patent 20 130 301 930, Nov. 14, 2013.
- [5] K. K. Parhi, *VLSI Signal Processing Systems: Design and Implementation*, 2007, John Wiley & Sons.
- [6] 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.
- [7] S.Y. Park and P.K. Meher, "Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic", in *Circuits and Systems - II: Express Briefs, IEEE Trans. on*, Vol.60, No. 6, pp. 346 - 350, Jun. 2013.
- [8] R. M. Hewlitt and E. S. Swartzlantler, "Canonical Signed Digit representation for FIR digital filters", in *Signal Processing Systems, 2000, IEEE Workshop on*, Lafayette, LA, 416 - 426, Oct. 2000.
- [9] K. Tsoumanis, N. Axelos, N. Moshopoulos, G. Zervakis and K. Pekmetzi, "Pre - encoded multipliers based on non - redundant radix - 4 signed - digit encoding", in *Computers, IEEE Trans. on*, Vol.65, No. 2, pp. 670 - 676, Feb. 2016.
- [10] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Transactions on A.S.S.P.*, Vol. ASSP-22, pp. 456-462, Dec. 1974.
- [11] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication", in *ACM Transactions on Algorithms (TALG)*, Vol. 3, No. 2, pp. 1-39, May 2007.
- [12] L. Aksoy, P. Flores and J. Monteiro, "Efficient design of FIR filters using hybrid multiple constant multiplication on FPGA", in *Computer Design (ICCD), 2014 32nd IEEE International Conference on.*, pp. 42 - 47, Oct. 2014.
- [13] A. Berkeman, V. Owall and M. Torkelson, "A low logic depth complex multiplier using distributed arithmetic," *IEEE J. Solid- State Circuits*, Vol. 35, No. 4, pp. 656-659, Apr. 2000.
- [14] E. O'Shea, "Bachet's problem: as few weights to weigh them all", *arXiv: 1010.5486*, pp. 1 - 15, Oct. 2008.
- [15] "IEEE Standard for binary floating - point arithmetic", *ANSI/IEEE754 - 1985*, 1985.
- [16] M. A. Basiri and N. M. Sk, "An efficient hardware - based higher radix floating point MAC design", *ACM Trans. on Design Automation of Electronic Systems*, Vol. 20, No. 1, pp. 15:1 - 15:25, Nov. 2014.
- [17] M. Sjalander and P. Larsson - Edefors, "Multiplication acceleration through twin precision", *IEEE Trans. on VLSI Systems*, Vol. 17, No. 9, pp. 1233-1246, Sep. 2009.
- [18] M. Vigliar and G. D. Licciardo, "Multiplierless coprocessor for difference of Gaussian (DOG) calculation," U.S. Patent 20 130 301 950, Nov. 14, 2013.
- [19] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers (sixth edition)*, Oxford University Press, 2008.
- [20] S. K. Park, "The r-complete partitions", *Discrete Mathematics*, No. 183, pp. 293-297, 1998.
- [21] Ø. J. Rødseth, "Enumeration of M-partitions", *Discrete Mathematics*, No. 306, pp. 694-698, 2006.
- [22] Virtex - 7 Family, DS183 (v1.23), Xilinx, San Jose, CA, USA, Jun. 23, 2015.
- [23] S. Arish and R. K. Sharma, "An efficient floating point multiplier design for high speed applications using Karatsuba algorithm and Urdvha - Tyriagbhyam algorithm", *Signal Processing and Communication (ICSC), 2015 International Conference on*, Noida, IND, pp. 303 - 308, Mar. 2015.
- [24] W. M. Chao and L. G. Chen, "Pyramid architecture for 3840x2160 QuadFull High Definition 30 frames/s video acquisition", *Circuits and Systems for Video Technology, IEEE Trans. on*, Vol. 20, No. 11, pp. 1499-1508, Nov. 2010.
- [25] M. Potkonjak and M. B. Srivastava, "Multiple Constant Multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, Vol. 15, No. 2, pp. 151-165, Feb. 1996.