IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 7, Issue 1, Ver. I (Jan. - Feb. 2017), PP 01-07
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

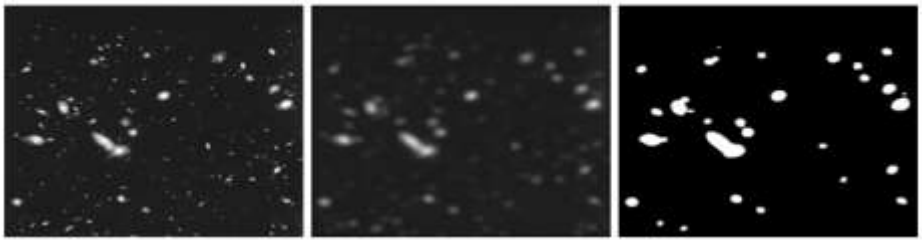
Optimized linear spatial filters implemented in FPGA

Ivan Kanev, Petya Pavlova
(Department CST, Technical University Sofia - Plovdiv branch, Bulgaria)

Abstract: *Linear spatial filters (LSF) are used for filtering of digital images with the purpose of blurring, noise reduction, detail enhancement etc. The realization of LSF confronts the capital problem of a lot of operations needed for their computation. In this paper, described is an approach for optimizing of LSF by utilizing parallel algorithms and their hardware implementation on FPGA. A model and an algorithm based on partial sums and aimed at calculating the filtered pixels are presented. Defined are criteria for comparing of the different types of linear filters. A schematic diagram of an FPGA-based DSP operational block is shown. VHDL is utilized for the hardware design. Conducted are studies focused on comparing the partial sums and the non-partial sums based methods of filtering. It is ascertained that the methods employing partial sums reduce the number of operations to the size of the window (3, 5, 7,...) . These FPGA-based LSF are suitable for applications using threshold detecting, edge detection or image detail enhancement.*
Keywords: *DSP, FPGA, Linear Spatial Filtering, Partial Sum, VHDL*

I. Introduction

The concept of filtering is related to the use of Fourier transform for signal processing in the frequency domain. The digital filtering of an image uses procedures which are applied directly on the pixels of the image. In this case, the term spatial filtering is employed to distinguish these processes from filtering in the frequency domain. In spatial filtering the value of the pixel being filtered is calculated by the cross-correlation or convolution operators applied on the pixel neighborhood [1]. Smoothing spatial filters are used for blurring and noise reduction in an image. Blurring is used in the image preprocessing. such as removing of small details in the image or overcoming small gaps in lines or curves (Fig. 1). Noise reduction is used to improve image quality. Spatial filtering are classified as linear and nonlinear. If the operations performed on neighbourhood pixels are linear, filters are classified as linear. Otherwise filters are classified as non-linear [2]. This article deals only with linear spatial filters (LSF), because they are the basis for important applications in digital filtering of images such as threshold detection, edge detection etc.



(a) (b) (c)

Fig. 1. Filter images LSF: (a) the image from the Hubble Space Telescope: (b) filtering the input image with a mask of dimensions 15 * 15; (c) performance thresholds [1].

A major problem in the implementation of LSF is that calculating the pixels being filtered requires multiple operations summation, multiplication and division. The number of these operations is increased proportionally to increase of the image size and reaches hundreds of millions. Oh that basis, they are difficult to apply \when employing general-purpose microprocessors (GPP), or DSP processors [3]. The limitations that are inherent of GPP and DSP processors can be overcome through implementing LSF on a single chip (System on a Chip (SoC)) based on Field Programmable Gate Arrays (FPGAs) [4]. FPGA is having the capability of parallel processing and hence it is a good platform for image processing [5]. Bailey, [6] analyzes various pipeline and parallel processing schemes of LSF, implemented on FPGA. The development of such schemes can be realized in two directions: new algorithms to calculate the pixel being filtered; reducing the number of operating units. Vasilev at al. [7] propose an algorithm to reduce the time to calculate operations with the use of partial sums. Khan [8], Malik [9] recommend FPGA-based LSF to be constructed with a DSP blocks using multiply accumulate (MAC). To optimize the number of registers used to store the weight coefficients a ROM- based finite-state machine (FSM) can be employed [10].

In this paper, discussed is an approach to optimize the LSF using parallel algorithms based on partial sums, and their hardware implementation on FPGA.

The paper is organized as follows. Section II presents a parallel algorithm and a DSP operating unit The results are analyzed in Section III. Section IV concludes the paper.

II. Parallel Algorithm And DSP Operational Block For Calculating LSF Using Partial Sums

The filtering of a pixel $p(x,y)$ in an image of size $M \times N$ results in a new pixel $g(x, y)$ with coordinates equal to the coordinates of the center of the neighbourhood and whose value is the result of the filtering operation [1]. The effect of filtering is determined by the mask of weight coefficients $w(a,b)$. The cross-correlation operator $G(x, y)$ used to calculate the filtered pixel employing linear spatial filters, and defined in the window of size $S \times S$, is presented by the following relation:

$$G(x,y) = \sum_{a=-k}^{k} \sum_{b=-k}^{k} w(a,b) \cdot p(x+a, y+b), \quad (1)$$

where

$$x = 0,1,2,..,M-1;\ y = 0,1,2,..,N-1. \quad (2)$$

$$k = \frac{S-1}{2}; \quad (3)$$

S is an odd number, and $S \geq 3$. (4)

The normalized value of the filtered pixel $g(x, y)$ can be calculated employing two methods:

Method # 1 - dividing $G(x, y)$ by the normalizing factor $NF(a,b)$

$$g(x,y) = G(x,y) / NF(a,b), \quad (5)$$

where

$$NF(a,b) = \sum_{a=-k}^{k} \sum_{b=-k}^{k} w(a,b). \quad (6)$$

Method # 2 - by multiplying the $G(x,y)$ by the reciprocal value of the normalization factor $NF^{-1}(a,b)$

$$g(x,y) = G(x,y).NF^{-1}(a,b). \quad (7)$$

Although both methods provide the same results, their implementation in FPGA-based LSF presents substantial differences. With Method # 1 the normalization is implemented through a divider synthesized with the FPGA logical elements, and with Method # 2 - through FPGA built-in multipliers. In this regard, method # 2 is preferable because it does not use the logical elements and the normalization is performed more quickly. [11]

1. Algorithm for parallel computation of the filtered pixels by using partial sums

The algorithm for the parallel calculation of $g(x,y)$ using the partial sums are based on the fact that the filtering of several consecutive windows includes pixels having common local coordinates. This circumstance can be used as a basis for algorithms in which, parallel to the calculation of the current value of $G(x,y)$ and $g(x,y)$, calculated are partial sums $PS1$ (x, y), $PS2$ (x, y), .. composed of pixels and weight coefficients that can be used in subsequent iterations.

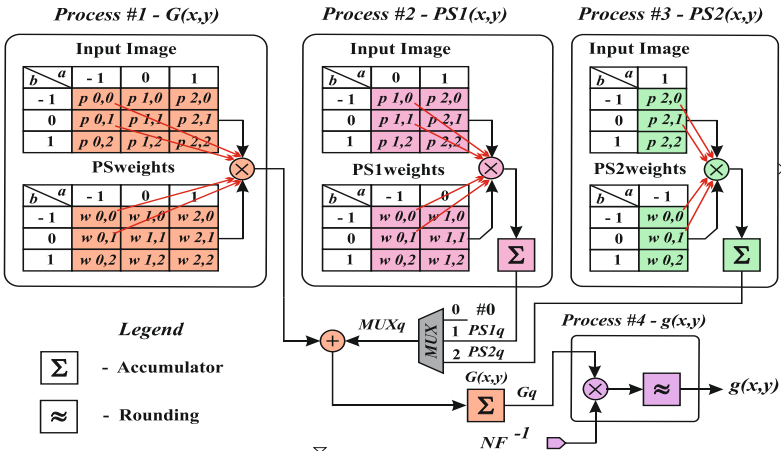


Fig 2. Conceptual model of a method for parallel calculation of LSF using partial sums: *Step* # 1*; x, y* = 1

In Fig. 2 shows a conceptual model of a method for parallel calculation of the LSF using the partial sums. In the proposed model it is assumed that the pixels of the input image p (x, y) arriving in succession at the input of the filter, by the columns of the window in which the filtering is carried out:
p(x,y) = p(0,0), p(0,1),.., p(3,0), p(3,1) p(3,2),p(4,0), p(4,1), p(4,2),..

The pixels of the input image are multiplied by the weight coefficients of the Gaussian kernels PS weights, PS1 weights, PS2 weights, and summed up in three accumulators (Σ). In the window of size $S = 3$ (4), four parallel processes are realized in three successive steps through which calculated are $G(x,y)$, $g(x,y)$, *PS1* (*x, y*), and *PS2* (*x, y*). The final values of the results are stored in the output registers *Gq*, *PS1q* and *PS2q* of the accumulator. The normalized value of the filtered pixel $g(x,y)$ is rounded by the factor $r, r \in \{0,1\}$, to ensure more accuracy to the calculations.

Start of the algorithm

y=0

Step # 1. This step is performed for all $x = 1$. Realized are three parallel processes in which calculated are $G(x,y)$, the final value of *PS1*(*x, y*), the initial value of *PS2*(*x, y*) and the current values registers *Gq*, *PS1q*, *PS2q* are set. To perform the calculations S^2 operations are required.

$x = 1;\ y = y + 1$, moving the window one position to the right

$$\textbf{\textit{Process \#1}}:\ G(x,y) = \sum_{a=-1}^{1} \sum_{b=-1}^{1} w(a+1,b+1)\,.\,p(x+a,y+b); \tag{8}$$

$$Gq \leftarrow G(x,y)\,. \tag{9}$$

$$\textbf{\textit{Process \#2}}:\ PS1(x,y) = \sum_{a=-1}^{0} \sum_{b=-1}^{1} w(a+1,b+1)\,.\,p(x+a+1,\ y+b); \tag{10}$$

$$PS1q \leftarrow PS1(x,y)\,. \tag{11}$$

$$\textbf{\textit{Process \#3}}:\ PS2(x,y) = \sum_{b=-1}^{1} w(0,b+1)\,.\,p(x+1,\ y+b)\,. \tag{12}$$

Step #2. This step is performed for $\forall x, x \neq 1$. Four parallel processes are implemented to estimate the current values and $G(x,y)$, the initial value of *PS1*(*x, y*) and the final value of the *PS2*(*x, y*). . To perform the calculations S operations are required.

$$\textbf{\textit{Process \#4}}:\ g(x,y) = Gq.NF^{-1} + r\,. \tag{13}$$

$x = x + 1$, moving the window one position to the right

$$\textbf{\textit{Process \#1:}}\ G(x,y) = PS1q + \sum_{b=-1}^{1} w(2,b+1)\,.\,p(x+1,\ y+b); \tag{14}$$

$$Gq \leftarrow G(x,y)\,. \tag{15}$$

$$\textbf{\textit{Process \#2:}}\ PS1(x,y) = \sum_{b=-1}^{1} w(2,b+1).p(x+1,\ y+b). \tag{16}$$

$$\textbf{\textit{Process \#3:}}\ PS2(x,y) = PS2(x-1,y) + \sum_{b=-1}^{1} w(2,b+1).\,p(x+1,\ y+b); \tag{17}$$

$$PS2q \leftarrow PS2(x,y)\,. \tag{18}$$

Step #3. This step is performed for. $\forall x, x \neq 1$. Implemented are four parallel processes through which current values and *G*(*x, y*), the final value *PS1*(*x, y*) and the initial value of *PS2*(*x, y*) are calculated. To perform the calculations S operations are required.

$$\textbf{\textit{Process \#4}}:\ g(x,y) = Gq.NF^{-1} + r\,. \tag{19}$$

$x = x + 1$, moving the window one position to the right

$$\textbf{\textit{Process \#1:}}\ G(x,y) = PS2q + \sum_{b=-1}^{1} w(2,b+1)\,.\,p(x+1,\ y+b); \tag{20}$$

$$Gq \leftarrow G(x,y)\,. \tag{21}$$

$$\textbf{\textit{Process \#2:}}\ PS1(x,y) = PS1(x-1,y) + \sum_{b=-1}^{1} w(2,b+1).p(x+1,\ y+b). \tag{22}$$

$$PS1q \leftarrow PS1(x,y). \tag{23}$$

$$\textbf{\textit{Process \#3:}}\ PS2(x,y) = \sum_{b=-1}^{1} w(2,b+1).p(x+1,\ y+b). \tag{24}$$

Operations for managing the cycles:
if $x \le M-2$ **then** *Step #2*
else if $y \le N-1$ **then** *Step #1*
else end of the algorithm

The timing of the steps and processes in the execution of the algorithm is shown in Fig. 3.

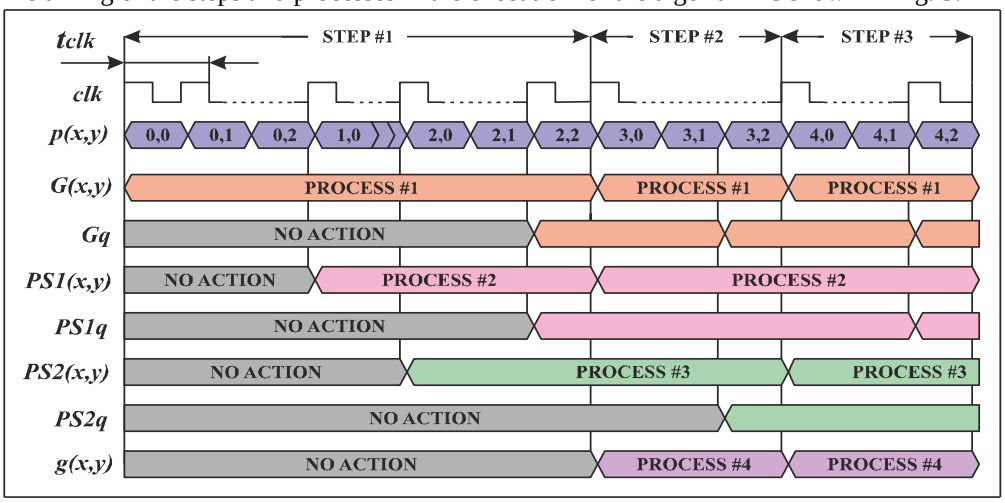


Fig. 3. Generalized timing model

The effect of applying the algorithm using partial sums can be assessed by comparing the algorithm without using partial sums. Let us, for a given k with the *Wn* denote the number of windows required to filter the input image of size $M \times N$, and with *S* the number of operations in a single window. Then

$$Wn = (M-k)\times(N-k). \tag{25}$$

The number of operations in algorithms without the use of partial sums O_C (8) and using my partial sums O_P (8) (14) (20) defined by the following expressions:

$$O_C = S^2.Wn + 1; \tag{26}$$

$$O_P = S^2(N-k) + S.Wn. \tag{27}$$

Then the relation

$$F_R = O_C / O_P, \tag{28}$$

can be used as a criterion for evaluating the maximum value of which reduces the number of operations in the algorithms using the partial sums as compared to the algorithms without using the partial sums.

The real effect of reducing the number of operations in the algorithms using the partial sums can be determined from the relation T_R:

$$T_R = T_C / T_P, \tag{29}$$

where:

$$T_C = tclk1.O_C; \tag{30}$$

$$T_P = tclk2.O_P; \tag{31}$$

$$tclk1, \tag{32}$$

is the period of clk (Fig. 3) in the algorithms without the use of partial sums;

$$tclk2, \tag{33}$$

is the period of clk in algorithms using partial sums.

In contrast to F_R, the relation T_R takes into account the influence of the period of the clk for which positive Slack is calculated. This factor is calculated after timing analysis of the hardware realization of filters.

2. FPGA-based DSP operational block

To realize the **a**lgorithm FPGA-based DSP **operational** block is developed (Fig.4). A combinational scheme in which parallel processes are combined with the use of pipelines is employed.

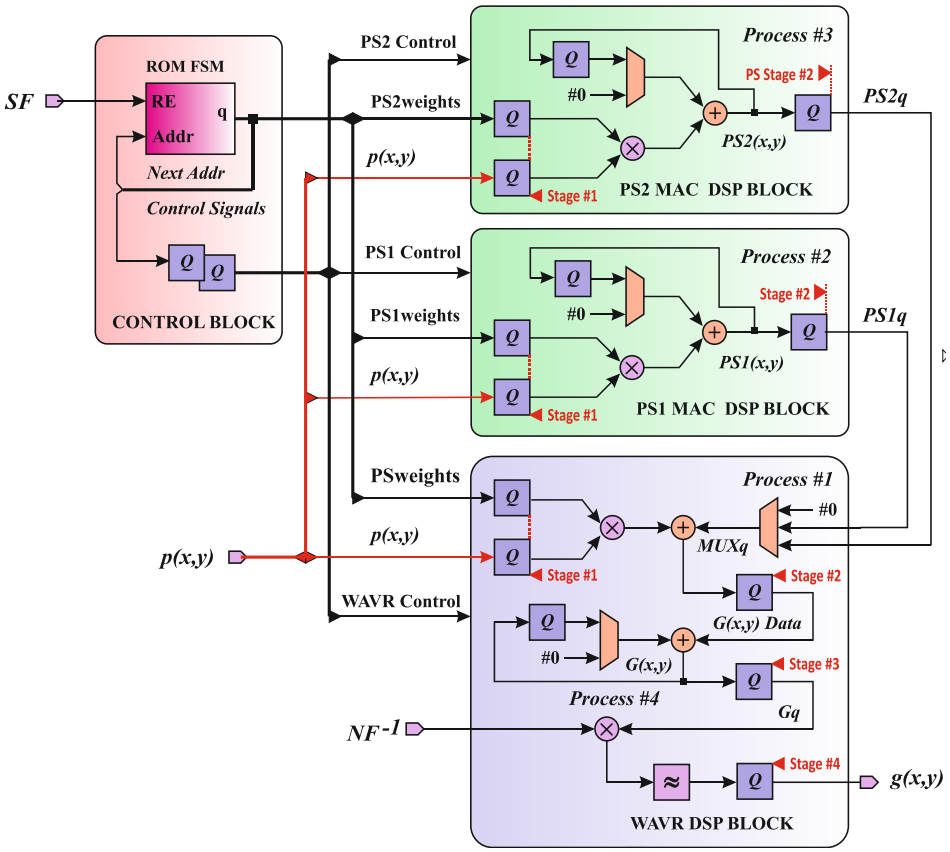


Fig. 4. Structural diagram of FPGA-based DSP operational block: $k = 1$.

DSP operational block consists of four sub blocks:

1. WAVR DSP BLOCK. In this block, calculate a normalized weighted value of the filtered pixel $g(x, y)$ (13), (19). For the realization of operations in this block is used a four stage pipeline.

2. PS1 (2) MAC DSP BLOCK. When $k = 1$, two identical multiply accumulate (MAC) DSP blocks are used to calculate the partial sums *PS1(x,y)* (10), (16), (22) и *PS2(x,y)* (12), (17), (24). The output registers *PS1q* and *PS2q* are set with the current values of *PS1(x,y)* and *PS2(x,y)* in compliance with equations (11), (23) and (18). For the realization of operations in these blocks a two stage pipeline is used.

3. CONTROL BLOCK. For managing and synchronizing the processes in DSP a finite state machine synthesized with the built-in FPGA ROM memory (ROM FSM) is used. This FSM generates three groups of signals:

3.1. Weight coefficients

Table I. Weight coefficients: *k=1*; *NF =16*

Weights/Step	Step #1									Step #2			Step #3		
PSweights	1	2	1	2	4	2	1	2	1	1	2	1	1	2	1
P1weights	NA			1	2	1	2	4	2	1	2	1	2	4	2
PS2weights	NA						1	2	1	2	4	2	1	2	1

3.2 Control signals. Signals WAVR Control, PS1 Control and PS2 Control is used to manage the registers and multiplexers in the individual DSP blocks.

3.3. Next Address. These signals are determined next state ROM FSM.

The interface of the DSP operational block is realized with the following input-output ports:

- Start filtering (SF) – input port. When this signal is set in high levels LSF filtering operations start.
- Input pixsel $p(x,y)$ – input port. Pixels of the input image are consecutively set at this port by the columns of the window in which the filtering is carried out.

- Reciprocal of normalization factor NF^{-1}- input port. Constant NF^{-1} (6) (7) is set at this port before the activation of SF and remains unchanged until of all the image pixels are filtered.
- *g(x,y)* – output port. At this port, the current value of the filtered pixel is set (7).

FPGA-based DSP operational block for parallel calculation of LSF using partial sums (Fig.4), can be extended to other values of k (3) with the instantiation of a new pair PS MAC DSP Block in the project, for each new value of k .

III. Result And Discussion

For the purposes of this study, VHDL is used for the hardware design. The low cost, low resources FPGA family Cyclone V of Altera is employed. Table II shows the utilized resources in the FPGA- based hardware implementation of a LSF

Table II. Resource utilization summary (Altera Cyclone VE 5CEBA4F17C6 Device)

Resource utilization	Available	Without partial sum		With partial sum	
		Used	Utilization	Used	Utilization
Logic utilization (in ALMs)	18480	14	0.0756 %	36	0.1948 %
Total registers	-	41	-	86	-
Total block memory bits	3153920	1792	0.0568 %	1984	0.0629 %
Total (Mult) DSP Blocks	66	2	3 %	4	6 %

Compared are the utilized resources with and without the use of partial sums. The relatively small share of the used adaptive logic modules (ALMs) is due to: the use of built-in FPGA DSP Multiplier Blocks; combining in a ROM-based FSM the signals for managing the operational block and weight coefficients.A static timing analysis between the related registers of the DSP operational block. The minimum values of the Slack calculated in the different processes, are shown in Table III.

Table III. Registers Path minimum slack summary

Process #	Registers Path	Clock Delay[1]	Clock Delay[2]	Data Delay	Slack
1	Gxy_Data to Gq	3.264	2..962	3.807	1.888
2	PS1weight to PS1q	3.262	2.652	4.627	0.81
3	PS2weight to PS2q	3.252	2.664	4.529	0.93
4	Gq to g	3.263	3.211	6.602	10.85

It is ascertained that the parameter Slack reaches critical values between registers PS1 (2) weight to PS1 (2) q (Fig. 5). The timing of Slack can be improved by increasing the number of steps of the pipelines through which PS1 (2) MAC DSP blocks are realized.

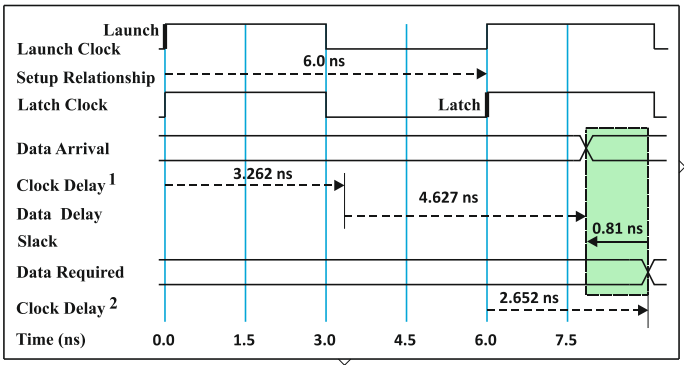


Fig. 5. Timing analisis registers to registers path: PS1weight to PS1q.

Table IV shows the dependence of *OC* (26), *OP* (27), *TC* (30), *TR* (31), *FR* (28), on the size of the image - M × N.

Table IV. Dependence of OC, OP , TC , TR , FR from M × N.

	M × N		
	640×480	1024×768	1600×1200
OC	2754730	7061770	16103529
OP	922554	2360826	5378634
TC [ms]	13,7736	35,3088	80,5176
TP [ms]	5,5353	14,1650	32,2718
FR	2,9860	2,9912	2.9940
TR	2,4883	2,4927	2,4950

Studies are carried out for k = 1 (3), tclk1 = 5ns (32) and tclk2 = 6ns (33). Although for the selected FPGA family positive Slack is reached for tclk2> tclk1, the relation *TR* reaches up to 83% of *FR*. As a result of studies carried out it is ascertained that with the increase of k increases the value of F_R, and $F_R \approx 2k+1$. The upward trend of F_R is shown in Fig. 6

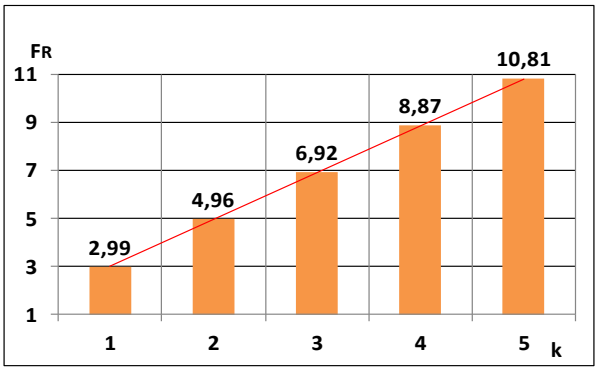


Fig. 6. Dependence of F_R on the size of the window k: M × N = 640 × 480

By increasing the size of the window a large number of parallel processes are realized, and this results in increased F_R.

IV. Conclusion

In this paper, discussed is an approach aimed at optimizing the LSF using parallel algorithms based on partial sums and their hardware implementation in FPGA. Presented is an FPGA-based DSP operational block to calculate the LSF. Provided is an option for expanding this block through instantiating new MAC DSP blocks within the module. Compared are the utilized FPGA resources with the use of partial sums, and without the use of partial sums. It is ascertained that for the hardware implementation of algorithms based on the partial sums a minimum number of resources is required. A static timing analysis among the registers of the DSP operational block is realized. The critical values of the parameter Slack are ascertained. Studied is the reduction of operations in the hardware implementation of the algorithm using partial amounts.

References

[1] Gonzalez R., Woods R., Digital Image Processing, third edition, Prentice Hall, 2012.
[2] Uwe Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays* (FPGA), *4th Edition*, Springer 2014.
[3] Jackson L., *Digital Filters and Signal Processing*, with MATLAB Exercises, *3rd Edition*, Springer 2010.
[4] Salunkhe A., Bombale U., Optimized Implementation of Edge Preserving Color Guided Filter for Video on FPGA, IOSR Journal of VLSI and Signal Processing , e-ISSN: 2319 – 4200, Volume 5, Issue 6, Ver. I (Nov -Dec. 2015), PP 27-33
[5] Bailey D., Design for Embedded Image Processing on FPGAs, First Edition, John Wiley & Sons (Asia) Pte Ltd., 2011
[6] Dimitrios B. et all, An FPGA-based hardware implementation of configurable pixel-level color image fusion, IEEE Trans. Geosci. Remote Sens., 50(2), 2012, 362–373.
[7] Vasilev N., Bosakova-Ardenska A., A Parallel Conveyer Algorithm for the Recursive Method of Scanning Mask for Primary Images Processing, CompSys Tech'2006, Veliko Tarnovo, Bulgaria, ISBN-13: 978-954-9641-46-2, pp IIIA.19-1 – IIIA.19-7, 2006.
[8] Khan S., Digital Design of Signal Processing Systems: A Practical Approach, First Edition. John Wiley & Sons, Ltd. Published 2011.
[9] Malik S. et. all, Implementation of MAC unit using booth multiplier & ripple carry adder, International Journal of Applied Engineering Research, Vol 7, No 11, 2012
[10] M. Rawski, H. Selvaraj, and T. Luba, An application of functional decomposition in ROM-based FSM implementation in FPGA devices, Journal of Systems Architecture, vol. 51, p. 424, 2005.
[11] Kanev I., Comparison of Two Methods for Computing FPGA-based Weighted Average Linear Spatial Filters CompSysTech'15, ACM International Conference Proceeding Series, Vol. 1008, ACM Inc., N.Y. USA, pages 276-283, 2015.