

# A FPGA Based Two Level Optimized Local Filter Design for High Speed Image Processing Applications

Majida Kazmi<sup>(✉)</sup>, Arshad Aziz, Pervez Akhtar, and Nassar Ikram

Department of Electrical Engineering (PNEC), National University of Sciences and Technology (NUST), H-12, Islamabad, Pakistan  
majida.kazmi@pnec.nust.edu.pk

**Abstract.** This work presents an efficient Field Programmable Gate Array (FPGA) based local filter design for portable and high speed image processing applications. It is highly optimized by using two level optimization. The first level optimization at design-level exploits tempo-spatial parallelism of filters by developing parallel/pipelined architecture. For exploiting spatial-parallelism, design computes partial results of multiple MACs in parallel and accumulates them via adder-tree for final result. Though it bears good performance aptitude but adder-tree incurs long critical path (4.713 ns) thus limits design performance. The critical path was reduced to 2.489 ns with temporal parallelism by pipelining the adder-tree. Design performance is further enhanced by deploying the second level optimization at post-implementation level where device aware floor-planning fine tunes the design. It aligns all utilized embedded resources of design on Xilinx Virtex-5 device and confines slice based logic across them. It results in packing the design within small area with reduced slice count and critical path (2.32 ns). After applying two levels of optimization, the design occupies 89 Slices, 3 DSP-Slices, 2 BRAM18 and achieves high frequency of 431.03 MHz.

**Keywords:** Local filters · Real time image processing · FPGA · XSG tool · Floor planning

## 1 Introduction

Digital image processing becomes a subject of common interest in emerging areas such as computer vision, medical, surveillance and industrial to name a few. In all of these applications, image processing operations help to improve the quality of images for its correct interpretation and analysis without human interference [1]. Most of these image processing operations are local by nature, in which calculation of an output pixel depends on a local neighborhood of input pixels. These operations are computationally expensive and time-consuming. Software based sequential architectures for such operations are relatively slow, thus not able to meet time constraints associated with real time systems [17].

Therefore to accelerate computation of these image processing operations, hardware such as Field Programmable Gate Array (FPGA) based parallel architectures have been explored [1, 6, 17].

Among different local image processing operations, one of the most frequently used operation is two dimensional convolution in local filters [4]. In the convolution operation, output pixel  $P_{out}$  is obtained by convolving a  $N \times N$  window of input pixels with the  $N \times N$  Filter Kernel which generally requires  $N^2$  multiplications and  $N^2-1$  additions. The quadratic growth of  $N$  factor implies that it is a computationally expensive and time consuming operation [6, 13, 17, 20]. The FPGA based parallel architectures exploit tempo-spatial parallelism of local filtering operation to accelerate its computation for real time applications [1, 6, 13, 16, 17, 20]. At the same time, due to its inherent re-configurability, it is also possible to modify the filter design at any stage to incorporate requirements of any specific application. For these reasons, the FPGA platform is viable for modeling the filter designs from its development phase till its final design.

Xilinx System Generator (XSG) is a tool for modeling and rapid prototyping of the image processing systems onto FPGA. It represents image processing system on Simulink/MATLAB platform and uses High Level Descriptions (HLD) to generate Verilog Hardware Description Language (HDL) code for FPGA implementation [16]. It also provides Hardware/Software co-simulation within its environment to check validity of hardware design on actual device at initial design stage. Ethernet and JTAG is been supported to communicate between hardware and Simulink for performing Hardware/Software co-simulation. Thus XSG provides an ideal user friendly environment for modeling of emerging image processing algorithms on FPGA [10, 11].

This paper focuses on an efficient implementation of local filters on FPGA by using XSG tool. Performance optimization of filter-design is achieved at two levels. At first level, Design space is explored on DSP friendly XSG environment to perform design-level optimization. It results in an efficient parallel and pipelined filter design that exploits tempo-spatial parallelism of local filters. Also computationally and memory intense portion of the design is efficiently implemented with DSP Slices and BRAM18 respectively. These powerful embedded FPGA resources [7, 8] perform their dedicated functions efficiently for filter design. At second level, optimization efforts are extended to post-implementation level where efficient floor-planning fine-tunes the design. It aligns utilized embedded resources on target FPGA device i.e. Virtex5 and packs associated Slices across them. It results in a compact design with high level performance gain. The proposed two level optimized local filter design is suitable for potable, high-speed real time image processing applications.

Rest of the paper is organized as follows. Section 2 discusses related work. In Sect. 3 explains our local filter design. Section 4 declares results and comparison. Conclusion is included in Sect. 5.

## 2 Related Work

We have gone through a brief literature review on FPGA based local filters for image processing using XSG. In [5] a parallel architecture of local filters is designed for MRI image filtering by using Xilinx Block Sets. They implemented nine basic filters of size  $5 \times 5$  by using Configurable Logic Blocks (CLBs). This architecture works at 228 MHz to process a  $64 \times 64$  image on Virtex-6 FPGA. Also in [18] local filters ( $3 \times 3$  Sobel,  $3 \times 3$  Prewitt and  $2 \times 2$  Robert) were designed for edge detection by using Xilinx Block Sets and implemented them on FPGA by using CLBs for a  $720 \times 480$  images. For a  $3 \times 3$  Sobel filter, their design works at 154.9 MHz and occupies 954 Slices and 5 BRAMs on Spartan 3A DSP XCSD3400A FPGA. Similarly, [3] has also designed different local filters and declared results for edge enhancement. Their design occupies 436 Slices on Virtex-6 FPGA. Instead of using Xilinx Block Sets, [14] used Black Box to design  $3 \times 3$  Sobel filter on XSG. The Black Box is used to incorporate HDL models into XSG. In their design, Black Box contains VHDL description for a  $3 \times 3$  Sobel filter. They implemented Sobel filter for edge detection on Spartan 3 A and Virtex-5 FPGA by using CLBs. Their design occupies 117 Slices at the working frequency of 54.5 MHz on Spartan 3 A and 103.4 MHz on Virtex-5 FPGA.

Instead of implementing local filters on FPGA by using only CLBs (i.e. Slices), it is also possible to exploit powerful embedded resource of FPGA i.e. DSP Slices in combination with CLBs. DSP Slice is dedicated to efficiently perform computationally intense arithmetic operations at high operating frequency as compared to the conventional CLBs based logic. In [15] Sobel filter is implemented for video processing by using combination of DSP Slices and CLBs. It occupies 1284 Slices and 4 DSP Slices on Spartan 3A DSP FPGA. Their design works at 68.4 MHz. However, [19] implemented Sobel and Gaussian filters by using only DSP Slices on Virtex-4 FPGA. For  $3 \times 3$  and  $5 \times 5$  filters, their design occupies 11 and 26 DSP Slices respectively at the working frequency of 100 MHz.

All of the above discussed reported works [3, 5, 14, 18] have implemented FPGA based local filters and performed design level optimizations. They implemented parallel architectures of local filters to accelerate their performance on FPGA. After design level optimization, [3, 14, 18] were not able to achieve significantly high operating frequencies for their design on low performance FPGA Spartan device. While [3, 5, 14, 18] selected high performance FPGA device (Virtex5 and Virtex6) for implementing local filters with design level optimization but still not achieved significant performance gain. In [15], powerful dedicated DSP Slices were utilized to perform computationally intense multiplication operation of filter at a high operating frequency, but the rest of their design logic occupied large number of logic Slices. Consequently the inefficiently mapped bigger design with long routing delays limits overall design performance. In [19] DSP Slices were used for implementing overall filter design with the aim of increasing speed and accuracy of design but not shown significant performance gain.

The above discussed implementations attempted to accelerate performance of local filters on FPGA by performing design level optimizations but none of them has shown any optimization effort at post-implementation level that would

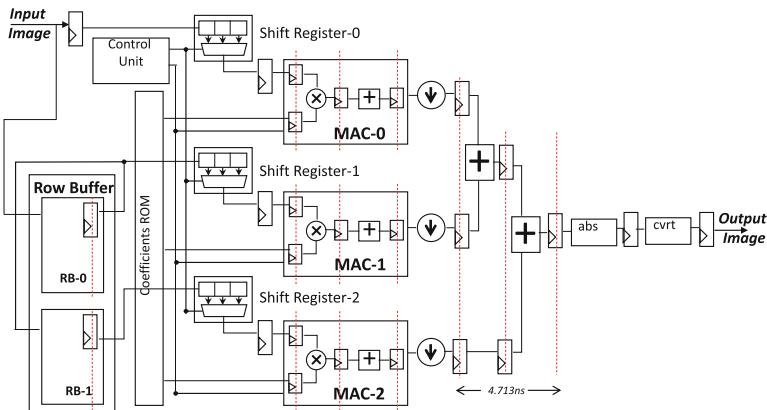
result in a compact local filter design with high performance gain. Therefore there is a need of an efficient design strategy which not only optimizes it at design level but also at post implementation level for portable and high speed image processing applications.

### 3 Our Work

In this work, we present an efficient implementation of local filters on FPGA by using XSG tool. A  $3 \times 3$  Gaussian filter (coefficients are 1,2,1;2,4,2;1,2,1) and a  $512 \times 512$  grey scale image of 8 bit precision has been chosen to explain our design concept. Figure 1 shows our design for a  $3 \times 3$  Gaussian filter. It comprises of Row Buffers (RBs), Shift Registers (SRs), Coefficients ROM, Multiply and Accumulate unit (MACs), down sampler, adder tree, absolute conversion unit (abs) and convert unit (cvrt).

The RBs temporarily store consecutive input image rows to provide the MACs a seamless flow of pixels per clock cycle via SRs. The MACs multiplies these input pixels with pre-stored filter coefficients and accumulate them to yield partial results. These partial results are down sampled and summed up by using down sampler and adder tree respectively. Absolute value is computed by abs, and then final result narrows down to the required level by cvrt.

The proposed filter design performs two levels of optimization i.e. design level and post implementation level which results in a compact filter design with high degree of performance gain. The functionality of design is also validated by using Hardware/Software co-simulation method to increase design productivity. Detailed implementation of filter design along with two levels optimization is explained in following sub sections.



**Fig. 1.** Our proposed  $3 \times 3$  filter design

### 3.1 Modeling of Local Filter Design Using XSG

We have used XSG for modeling of our filter design onto FPGA. The complete XSG model of our filter design is shown in Fig. 2. It comprises of four main units i.e. Input Unit, Internal Memory Unit, Arithmetic Unit and Output Unit. The Input Unit acquires a  $512 \times 512$  input image and delivers pixels array to Internal Memory Unit. The Internal Memory Unit stores 2 consecutive rows of input image on FPGA by using 2 RBs. These RBs provide multiple pixels per clock cycle to Arithmetic Unit for parallel computation. The Arithmetic Unit performs convolution operation on input pixels and outputs filtered image pixels. The array of filtered pixels is converted back into  $512 \times 512$  image matrix by using Output Unit. Detailed working of each unit is discussed separately in next sub-sections.

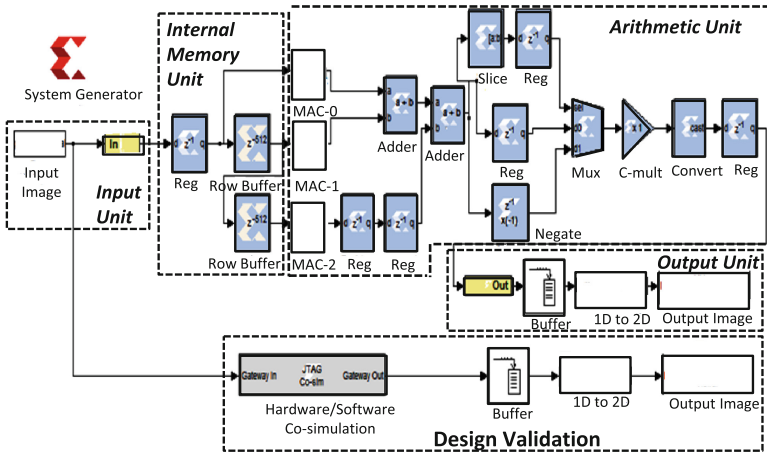


Fig. 2. Filter design modeled on Xilinx system generator

**Input Unit.** The Input Unit acquires a  $512 \times 512$  grey scale image and behaviorally reshapes it from (row  $\times$  column) matrix into (time  $\times$  sample) matrix. The reshaped input image data has the time steps in first column and the pixels stream in second column. Since the XSG is a time based tool, thus the time step variable is implicitly considered by the input image. The Gateway In block of Input Unit converts double precision input image pixel data into a fixed point type and serves as an input port for the top level module of our HDL design.

**Internal Memory Unit.** Input image data via Input Unit is buffered in Internal Memory Unit. The 2 RBs in this unit are implemented with 2 BRAM18 primitives in simple dual port configuration. Each BRAM18 stores one complete row of  $512 \times 512$  input image. With intake of single input pixel per clock cycle from external memory, the Internal Memory Unit provides multiple pixels per

clock cycle to parallel filter design for output pixels calculation at the throughput rate of 1clock/pixel.

**Arithmetic Unit.** Arithmetic unit is the main unit of our design which performs filtering operation (i.e. 2D convolution) on input image pixels. It is a parallel and pipelined architecture which exploits tempo-spatial parallelism of local filters. It comprises of three parallel connected 3tap MAC units, adders, absolute conversion unit (abs), convert unit and pipeline registers as shown in Fig. 2. Each MAC unit performs 1D convolution operation. It consists of DSP Slice, Op-mode, Addressable Shift Register (ASR), Distributed RAM (DRAM), Comparator, Counter, Capture Registers, Down Sampler and pipeline registers as shown in Fig. 3. The ASR implements SRs to shift input pixels over filter window (coefficients) for convolution operation. The filter coefficients are pre-stored in a DRAM implemented with FPGA Slices. A Comparator and a counter are required to implement address logic for ASR and DRAM. Counter generates addresses and Comparator creates the reset and enables signals for ASR and DRAM. The signal is asserted at 0 addresses and is delayed for pipeline stages. The DSP Slice is utilized to perform multiplication and accumulation operation upon getting instruction ( $P=P+(A*B)$ ) by Op-mode. Output of DSP Slice is fed to a capture register that captures its final result after accumulating full set of filter samples. As sampling period of MAC is higher than desired output sampling period, so output from capture register is fed to a down sampler. It's down sampling rate is equal to the length of filter's coefficient array. For our case study i.e. a  $3 \times 3$  filter, coefficient array length is 3 so the down sampling rate is also equal to 3.

Down sampled partial result from MAC units are then added to get a final output value. This is done by using a 2 stage adder tree structure. The absolute value of output is computed by using abs which is comprised of slice, negate and multiplexer (mux) blocks. Due to additions of partial results of parallel connected MAC units, the output value of adder tree is a larger bit number as compared to input. These results are rounded and quantized back to desired level. We used bit width convertor (convert) to narrow down final output to desired bit width. As we used 8 bit gray scale images at input, so our desired output level is set to 8

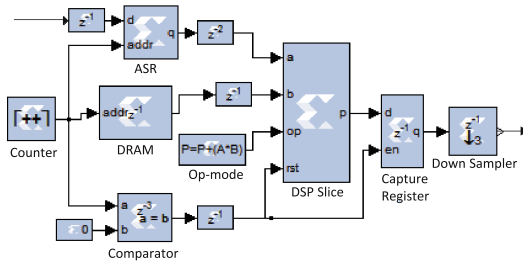


Fig. 3. MAC unit of our filter design

bit. The 8 bit filtered pixel of Arithmetic Unit in fixed point format is delivered to Output Unit of design.

**Output Unit.** The Output Unit acquires filtered output pixels from Arithmetic Unit. Its Gateway Out block converts fixed point output data to Simulink double precision value and serve as an output port for the top level module of the HDL design. The 1D output data stream then needs to convert back into 2D image matrix for display. A buffer is employed to convert scalar samples to frame at lower sampling rate, then conversion of 1D data into 2D image matrix is performed by using 1D to 2D conversion block in order to acquire output filtered image.

### 3.2 First Level Design Optimization

In first level of optimization, we explore design space of chosen local filter to exploit its inherent tempo-spatial parallelism. Firstly, for exploiting spatial parallelism, we design a parallel filter architecture in which the 2D filter operation (2D convolution) is partitioned into 3 parallel 1D filter operations (1D convolution). The 1D filter operation is performed simultaneously on input pixels of 3 rows to compute partial results. These partial results are then added up to the computation of final result.

As shown in Fig. 1, the input image pixels data from an external memory enters into RBs. For a  $3 \times 3$  filter, 2 consecutive rows of input image are stored in 2 RBs implemented with 2 BRAM18 primitives (18 Kb BRAM). The use of RBs provide multiple pixels per clock cycle required by parallel filter design to exploit its spatial parallelism [9]. The RBs introduces an initial delay to our design which is equal to the time needed until the first 2 rows of the image are stored in RB. Subsequently, they provide required pixels per clock cycle to our filter design without any further delays. Pixels from RBs enter in SRs which are used to shift them over filter windows.

Input pixels via RBs and SRs, goes into 3 parallel connected 1D filters. The MAC units perform 1D filtering operation. Each MAC multiplies 3 pixels with 3 filter coefficients and accumulates them to give a single output pixel (i.e. partial result). Since MAC is the most computational intense unit of overall design, it is implemented with embedded DSP Slice which is a dedicated FPGA resource for performing high-speed arithmetic operations. The DSP Slice efficiently performs MAC operation at high operating frequency, low power consumption and high accuracy as compared to conventional Slice based MAC unit.

The output of MAC is sent to a down sampler unit (with down sampling rate equal to the length of filter's coefficient array i.e. 3 for said case study). Down sampled data of MAC units is accumulated by a  $(N-1)$  stage adder tree (i.e.  $3-1=2$  for said case study). It accumulates all inputs and gives a filtered output pixel (i.e. final result). Absolute value is computed by abs, and then final result narrows down to required level by cvrt.

This parallel filter design bears a good performance aptitude however, the deployed adder tree structure incurs long critical path of 4.30 ns thus limiting

the overall design performance. To cater this bottleneck, we exploit temporal parallelism of adder tree structure in which its long combinational path is divided into smaller paths by inserting pipeline registers. These pipeline registers break long critical path associated with a combinational path of adder structure into small critical paths.

Critical paths associated with inputs and outputs of utilized embedded resources i.e. BRAM18 and DSP Slices in our design are also reduced by using their embedded primitive registers. These primitive registers hold the data of BRAM/DSP Slice in sub-pipeline manner to break longer critical path into shorter critical paths. We used output primitive registers of the BRAM and also multiplier primitive register of the DSP Slice. As these registers are embedded within these resources, they enhance overall performance of our design at no overhead hardware cost. By inserting pipeline and sub-pipeline stages in the design, its critical path is reduced to 2.48 ns thus increasing the overall performance (operating frequency) of design.

### 3.3 Second Level Design Optimization

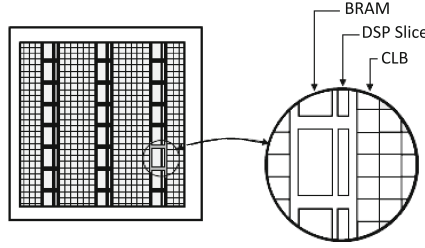
The first level optimized design was modeled on XSG and then imported onto Xilinx ISE Environment. From here, we extend our optimization efforts for second level of optimization, in which we fine-tune the design by its device aware floor planning. Xilinx Synthesis Tool (XST) place and route our design on target Virtex5 device by using conventional Computer-Aided Design (CAD) mapping algorithms. These mapping algorithms are heuristic in nature that shows good results but not guarantee the optimal solution [12]. Therefore for optimal solution, besides using XST efficiently by choosing optimal mapping strategy for our design, we perform device aware manual floor planning of our design on target Xilinx Virtex5 device.

Our design is comprised of dedicated resources (BRAM18 and DSP Slice) in combination with CLBs (i.e. logic Slices). Both of the dedicated resources are available on target Virtex5 FPGA device in separate columns adjacent to each other as shown in Fig. 4. Also, pitch of BRAM tile (comprising two BRAM18 primitives) matches the DSP tile (comprising two DSP Slices i.e. DSP48E) [8]. We manually map both embedded resources of our design on FPGA such that the design creates a natural alignment due to alike tile pitches. Afterwards we pack the associated Slice based logic across utilized dedicated resources as shown in Fig. 5. This alignment of dedicated resources on device and placement of logic closely across these dedicated resources confined the overall design to a small device area. It not only results in further reducing critical path of confined design to 2.32 ns but also reduces occupied Slices of our design with reduced interconnect usage.

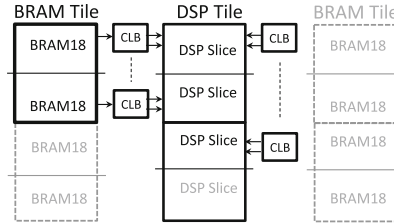
### 3.4 Design Validation

The functionality of our filter design is validated to improve its productivity for real time systems. We use Hardware/Software co-simulation to provide the





**Fig. 4.** Xilinx Virtex5 architecture



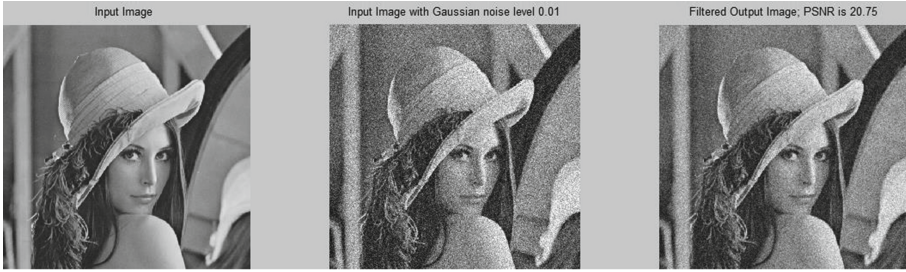
**Fig. 5.** Placement of our filter design on Virtex5

hardware version of design in flexible simulink environment. This hardware version interacts with the underlying FPGA device during simulation, performs necessary device configuration, data transfer and clocking. The JTAG cable is used to communicate between FPGA device and Simulink. The bit stream generated is verified by downloading it onto the FPGA board and ensuring that it runs successfully.

## 4 Results and Discussion

This work efficiently implements local filters on FPGA device using XSG. A  $512 \times 512$  grey scale image and a  $3 \times 3$  Gaussian filter were considered as a case study. The design was tested and evaluated on Virtex-5 (Xc5vfx100tff1738-3) FPGA device within the ISE 13.1 development Suite. The test case filtration on a standard Lena image by using our design is shown in Fig. 6.

Table 1 shows implementation results of our filter design in terms of device utilization and operating frequency. The design after first levels optimization utilizes 2 BRAM18, 3 DSP Slices and 145 Slices at operating frequency of 411.18 MHz. The results are improved by performing second level optimization. After second level optimization, Slice count of design is reduced by 38.6% with a 5% performance gain. The design after two level of optimization fits in just 89 Slices, 2 BRAM18 and 3 DSP Slices and achieves high operating frequency of 431.03 MHz. Since the filter is implemented in a parallel and pipelined way, after an initial latency it produces an output at every clock cycle. Therefore the design at this frequency is capable to sustain a throughput rate of 431.03 Mega Operations



**Fig. 6.** Test case Gaussian Filtering on a standard Lena input image

**Table 1.** Implementation results for said case study on Virtex5 (Xc5vfx100tf1738-3).

FPGA resources	First level optimization	Second level optimization	% Improvement
Slices	145	89	38.6 %
DSP Slice	3	3	0
BRAM18	2	2	0
Frequency (MHz)	411.18	431.13	5 %

per Second (MOS). With this throughput, for high definition image resolution  $1920 \times 1080$ , the architecture performs at 208 frames per second far exceeds the real time image processing requirements.

Comparison of our results with other reported results [15, 18, 19] is shown in Table 2. Since these results were available on different FPGA devices, therefore for a fair comparison we provide results in terms of equivalent slice count and equivalent frequency which is calculated by multiplying reported results with a normalizing factor. Firstly, the normalizing factor for slice count is calculated on basis of the equivalent logic which can be synthesized by the logic resources of different Xilinx FPGA devices. One Slice of Virtex-4 and below families is comprised of two LUT4 to perform 32 lookup operations while One Slice of Virtex-5 and above families is comprised of four LUT-6 to perform 256 lookup operations therefore a factor of 8 ( $256/32$ ) is applied as normalizing factor for their comparison [2] i.e. 1 Slice of Virtex-5 families or above is equivalent to 8 Slices of Virtex-4 or below families. Secondly the normalizing factor for frequency is also calculated by taking ratio of maximum achievable frequency of two dissimilar FPGA devices under comparison. The maximum achievable frequency of these designs on Spartan 3, Virtex-4, Virtex-5, and Virtex-6 are 320 MHz, 500 MHz, 550 MHz and 600 MHz therefore the frequency normalizing factors with respect to target Virtex-5 device are 1.71, 1.1 and 0.9 respectively. Moreover, besides considering the equivalent slice count and equivalent frequency for eliminating the impact of technology difference on results, the size of all reported designs are also scaled to a  $3 \times 3$  size for eliminating difference in results due to different design sizes. The results of scaled design in terms of equivalent Slice count

**Table 2.** Results comparison.

Work	Device	Equivalent frequency (MHz)	Equivalent slices	DSP slices
[15]	Spartan3	117.5	81	4
[18]	Spartan3A DSP	266.2	60	b
[19]	Virtex4	110	a	11
[5]	Virtex6	210.8	a	b
Proposed	Virtex5	<b>431.13</b>	<b>89</b>	3

<sup>a</sup>Not Given; <sup>b</sup>Not Used

and equivalent frequency is compared in Table 2. It is evident that our design outperforms all other implemented designs in terms of operating frequency and at the same time occupies reduced FPGA resources.

## 5 Conclusion

This paper presented an efficient implementation of local filters on FPGA. Performance optimization of filter-design is achieved at two levels. Firstly at the design level by exploiting tempo-spatial parallelism and secondly at post-implementation level by efficient device aware floor-planning which further fine-tunes the design and confine the design across dedicated embedded resources. The proposed two level optimized local filter design with low resource consumption and high operating frequency is a suitable solution for portable, high-speed ( $1920 \times 1080@208$ ) real time applications in the field of computer vision, medicine, military and industry to name a few.

## References

1. Bailey, D.G.: Design for Embedded Image Processing on FPGAs. John Wiley & Sons, Singapore (2011)
2. Betz, V., Rose, J., Marquardt, A.: Architecture and CAD for Deep-Submicron FPGAs, vol. 497. Springer Science & Business Media, New York (2012)
3. Elamaran, V., Praveen, A., Reddy, M.S., Aditya, L.V., Suman, K.: FPGA implementation of spatial image filters using Xilinx system generator. Proc. Eng. **38**, 2244–2249 (2012)
4. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice Hall, Upper Saddle River (2002)
5. Hasan, S., Yakovlev, A., Boussakta, S.: Performance efficient FPGA implementation of parallel 2-D MRI image filtering algorithms using Xilinx system generator. In: 2010 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP), pp. 765–769. IEEE (2010)
6. Hedberg, H.: Image processing architectures for binary morphology and labeling. Lund University (2008)
7. Xilinx Inc.: Virtex-5 FPGA xtremedsp design considerations user guide, January 2009

8. Xilinx Inc.: Virtex-5 FPGA User Guide v5.4 (2012)
9. Kazmi, M., Aziz, A., Akhtar, P., Kundi, D.E.S.: FPGA based compact and efficient full image buffering for neighborhood operations. *Adv. Electr. Comput. Eng.* **15**(1), 95–104 (2015)
10. Kiran, M., War, K.M., Kuan, L.M., Meng, L.K., Kin, L.W.: Implementing image processing algorithms using hardware in the loop approach for Xilinx FPGA. In: *International Conference on Electronic Design, ICED 2008*, pp. 1–6. IEEE (2008)
11. Moreo, A.T., Lorente, P.N., Valles, F.S., Muro, J.S., Andrs, C.F.: Experiences on developing computer vision hardware algorithms using Xilinx system generator. *Microprocess. Microsyst.* **29**(8), 411–419 (2005)
12. Oklobdzija, V.G.: *The Computer Engineering Handbook*. CRC Press, Boca Raton (2001)
13. Perri, S., Lanuzza, M., Corsonello, P., Cocorullo, G.: A high-performance fully reconfigurable FPGA-based 2D convolution processor. *Microprocess. Microsyst.* **29**(8), 381–391 (2005)
14. Said, Y., Saidani, T., Smach, F., Atri, M.: Real time hardware co-simulation of edge detection for video processing system. In: *2012 16th IEEE Mediterranean Electrotechnical Conference (MELECON)*, pp. 852–855. IEEE (2012)
15. Said, Y., Saidani, T., Smach, F., Atri, M., Snoussi, H.: Embedded real-time video processing system on FPGA. In: Elmoataz, A., Mammass, D., Lezoray, O., Nouboud, F., Aboutajdine, D. (eds.) *ICISP 2012*. LNCS, vol. 7340, pp. 85–92. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-31254-0\\_10](https://doi.org/10.1007/978-3-642-31254-0_10)
16. Saidani, T., Atri, M., Dia, D., Tourki, R.: Using Xilinx system generator for real time hardware co-simulation of video processing system. In: Ao, S.-L., Gelman, L. (eds.) *Electronic Engineering and Computing Technology*. LNEE, vol. 60, pp. 227–236. Springer, Netherlands (2010)
17. Samarawickrama, M.G.: *Performance Evaluation of Vision Algorithms on FPGA*. Universal-Publishers (2010)
18. Sudeep, K., Majumdar, J.: A novel architecture for real time implementation of edge detectors on FPGA. *Int. J. Comput. Sci. Issues* **8**(1), 193–202 (2011)
19. Wasfy, W., Zheng, H.: General structure design for fast image processing algorithms based upon FPGA DSP slice. *Phys. Proc.* **33**, 690–697 (2012)
20. Woods, R., McAllister, J., Lightbody, G., Yi, Y.: *FPGA-Based Implementation of Signal Processing Systems*. Wiley Online Library (2008)