High Performance Timing-Driven Rank Filter

Péter Szántó, Béla Fehér
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
{szanto, feher}@mit.bme.hu

Gábor Szedő
Xilinx Inc.
2100. Logic Dr, San Jose, CA 95124, USA
San Jose, California, USA
gabor.szedo@xilinx.com

***Abstract*—This paper presents an FPGA implementation of a high performance rank filter for video and image processing. The architecture exploits the features of current FPGAs and offers tradeoff between complexity and clock speed. By maximizing the operating frequency the complexity of the filter structure can be considerably reduced compared to previous 2D architectures.**

I. INTRODUCTION

Rank order filtering is a non-linear filtering technique, which selects an element from an ordered list of TAP number of samples. In the two-dimensional (2D) case filtering takes place on the contents of a rectangular window (or more generally, an arbitrary shape), which slides across the image. Every time the window is moved by one pixel column, a set of obsolete elements are discarded and a set of new elements are inserted. The samples within the window are sorted and the element with the specified rank replaces the output element of the window. Most typical ranks are median, minimum and maximum, but the selection can be easily tailored to the needs of any application. Compared to other filters, such as FIR, Laplacian or blur filters, rank filters can effectively remove impulse like noises while preserving the edges of the original image. This can be very useful for various applications, for instance removing certain types of transmission noises, or pre-processing for edge detection. This paper presents a hardware architecture that is tailored for high performance color video processing but can be used in various applications as an IP block by taking advantage of the design time parameterization. The paper concentrates on the timing-driven architecture selection which exploits the high operating frequency of recent FPGAs, thus reduces hardware resource requirements.

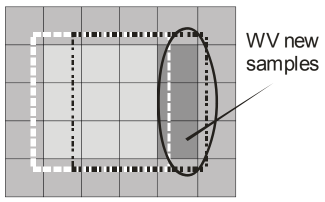


Figure 1. 2D image filtering

II. PREVIOUS WORK

The successful adaptation of rank filters in different applications catalyzed research activities for new algorithms and implementations.

Bit serial approaches [1], [2] provide the lowest complexity, but do not lend themselves well for high sample rate implementations, as filtering performance is proportional with the precision of the input data. However, the processing rate typically does not depend on the number of samples which change between processing cycles.

Insert-delete or sorting network based architectures [3] explicitly orders the incoming samples. In every cycle, the least recent sample is discarded and the most recent input is inserted into the magnitude sorting structure at the appropriate location. While these solutions require relatively few comparators, the feedback nature of the algorithm hinders pipelining.

Another set of applications store the samples in the order of arrival and select the appropriate output sample by calculating the location of the output sample dynamically. These architectures are easier to pipeline and still require few comparators.

III. PROPOSED ARCHITECTURE

When filtering images or video, the filter window slides horizontally on the input image, just as Fig. 1 shows. In case of a simple rectangular window, to generate a valid output, *WV* (vertical size of the filter window) new input samples should be processed. Therefore, for non bit serial implementations, an important classification criterion is the level of input parallelization.

Word-serial architectures can process one input sample per clock cycle. This is the typical structure for filtering 1D inputs, but it is also applicable for 2D filtering. In this case the filter should operate at *WV* times of the input pixel frequency and generates a valid input sample every WV^{th} clock cycle.

The other extremity is the full-parallel approach – these filters can generate valid output every clock cycle, irrespectively of the number of input samples required to achieve this. Consequently, such filters process *WV* new samples in a single clock cycle. Hence the required operating frequency equals to the input pixel frequency, while at the same time hardware resource requirements are greatly increased. Previous papers typically considered fully parallel architectures as 2D filters, however, as this paper proves, using recent FPGA technologies this solution is sub-optimal due to the inefficient resource utilization.

Multi-word architectures are hybrid solutions: in one cycle they can handle more than one input samples, but less then the fully parallel implementation (from now on, let *NI* denote the number of new input samples in a single cycle). This solution allows finding a good balance between operating frequency and hardware complexity. Using a given filter window and input pixel frequency, the required operating frequency can be computed:

$$FO_{\max} = FS\frac{WV}{NI} \tag{1}$$

When processing color images using the full per-pixel information (e.g. full RGB or YCbCr values) is not a convenient solution. Filtering these components independently not only increases computational requirements, but may introduce blur effects, as it may generate new color values which were non-existent on the input image. A better solution is to use a magnitude-like value, e.g. luminosity. If the input format does not contain such a component, it can be generated within the filter.

A. *Global Filter Architecture*

The proposed architecture consists of five main components (illustrated on Fig. 2): the Line Buffer, the optional Filter Value Generator (FVG), the Delay Line, the Filter Core and the Control Unit (CNTRL). The Line Buffer stores *WV-1* lines of the original input frame in internal memory. The Filter Value Generator is only required if the input format does not contain magnitude-like component, for YCbCr or YUV input representations this module can be omitted as the Y component lends itself well for magnitude ordering. For RGB input, luminance, a typical magnitude value can be calculated.

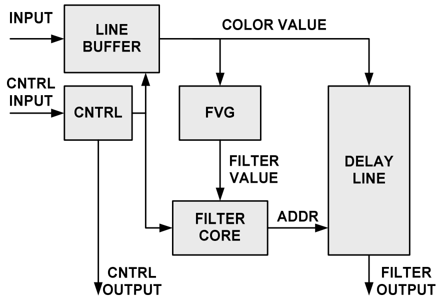


Figure 2. Global Filter Architecture

The Delay Line is an addressable FIFO which stores the full per-pixel information of the pixels residing inside the Filter Core. The Filter Core itself uses the values generated by the FVG and generates the appropriate address for the Delay Line. The Control Unit generates properly delayed synchronization signals and output valid signals.

Henceforward only the Filter Core and its extensions are discussed in details, as this is the essential part of the filter.

B. *Word-serial Filter Core*

The operation of the Filter Core is based on observations introduced in [5]. As a first assumption the filter contains TAP number of different samples. For each sample, an index value is generated, which equals to the number of samples which are smaller than the given sample. This results in TAP distinct values for the TAP samples, which range from 0 (smallest sample) to TAP-1 (largest sample). The ranked sample is the one which has the index value equal to the required rank. The block diagram on Fig. 3 illustrates the hardware implementation of the algorithm for TAP=5.

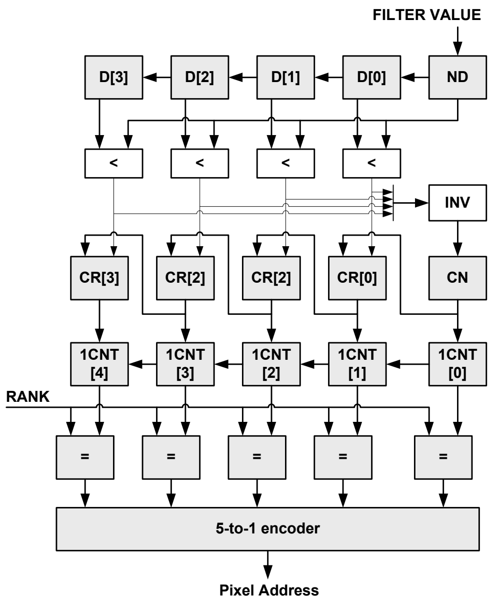


Figure 3. Filter Core

The *D[3:0]* data registers store older filter values, while the new data value is saved into the *ND* register. In every cycle, these registers shift their data to the left. Older values are compared with the new value (result is '1' if the new value is smaller than the older, '0' otherwise), and the comparison result is saved into the LSB position of TAP-1, TAP wide registers (*CR[3:0]*). The MSB positions of these *CR* registers are updated with the value of the previous *CR* register, so:

$$CR[k] = \{ CR[k-1](TAP-2:0), C[k]\} \tag{4}$$

where (:) denotes bit selection, {} denotes concatenation and C[k] denotes the kth comparison result. Expressively, the comparison result of a given value moves to the left together with the filter value. Therefore, at any given time, *CR[k]* stores the comparison results of *D[k]* with all the other values within the filter. The TAP wide register for the new value (*CN*) is computed differently: it is generated using the negated result of the comparators – namely, the kth bit is updated with the (k+1)th comparison result. The 0th bit (self-comparison) is set to '0'. Counting the '1's in the *CR[]* and *CN* registers gives the number of values which are smaller than the given value. This bit summing operation is done by the 1CNT modules. The straightforward way is to use an adder tree with TAP one bit inputs; for the CN register this is the only solution, as its content can change completely from clock to clock. The *CR[]* registers, however, offers some optimization possibilities. When generating *CR[k]*, only two bits changes from *CR[k-1]*: the MSB (comparison result with the discarded sample) and the LSB (comparison result with the new value). Therefore, bit summing can be implemented using an incrementer/decrementer. The result of the bit summing blocks are compared with the required rank, thus generating a TAP bit result containing exactly one '1' at the pixel position of filter window which contains the required output. An encoder passes this position to the Delay Line as an address. Table I shows an example with the data registers (*D[]*, *ND*), *CR[]*, *CN* and the output of the 1CNT blocks.

TABLE I. FILTERING EXAMPLE

	[3]	[2]	[1]	[0]	new
D[ ], ND	0	25	37	12	12
CR[ ], CN	00000	10011	11011	10000	10010
1CNT[ ]	0	3	4	1	2

C. Multi-word Filter Core

The architecture presented in the previous section can be easily extended to be able to process more than one new filter values per clock cycle. Instead of one data position, the data registers (*D[]*) and the comparator result shift registers (*CR[]*) should shift by *NI*. The yet single *CN* and *CR* registers become register arrays with *NI* elements. The number of comparators is increased, as all old samples should be compared with all new samples and new samples should be compared with each other. The required number of comparators for a *TAP* sized filter with *NI* new samples:

$$C = (TAP - NI) * NI + \frac{NI * (NI - 1)}{2} \tag{5}$$

If *WV* is not an integer multiply of *NI*, the bandwidth of the filter core input supersedes that of the input stream, so in some clock cycles the number of valid new data is going to be less than *NI*. The simplest solution to make the filter capable of processing different number of new samples is to insert multiplexers into the appropriate data paths, in front of *D[]*, *ND[]*, *CR[]* and *CN[]* registers. Two-to-one multiplexers are sufficient, because during the operation of the filter there are only two different scenarios. Either all *NI* inputs are valid, or there are only (WV mod NI) legal values (see Fig. 4). Thus the size of multiplexers is limited to 2:1, but still a numerous multiplexers are required.

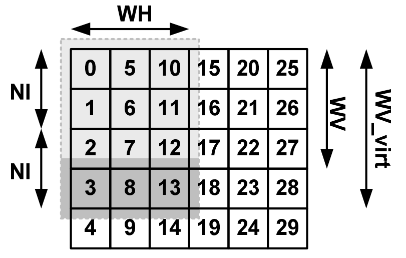


Figure 4. Virtual filter kernel

Another solution is to insert padding samples as necessary, so in every clock cycle *NI* new samples are entered, thus creating a virtual filter (from now referred as virtual filter kernel). Fig. 4 illustrates such kernel for the *WV*=3, *NI*=2 case. Valid samples in the window are marked with light grey; padding samples are marked with dark grey (the actual value of the padding samples is irrelevant). Obviously, this method makes the virtual kernel size larger than the real filter window, hence requires more hardware resources, as parts of the Filter Core scales with the virtual kernel size. Fig. 5 presents the contents of the data registers clock by clock – using the numbers on Fig. 4 – as new inputs are inserted and the filter window is moved horizontally. Valid and invalid (padding) samples are marked just as on the previous figure. Samples on the right are the input samples. As most of the data registers contain both valid and invalid samples during operation, comparisons are done using all required data registers, irrespectively of the validity of the actual sample. As a result the number of comparators required scales with the size of the virtual filter kernel. Padding samples are masked after the comparator result registers (*CR[]*, *CN[]*), but before the 1CNT blocks. For each older sample, masking is done on 2**NI* bits: *NI* bits mask the comparison results with the *NI* new samples, and another *NI* bits mask the comparison results of the oldest *NI* samples.

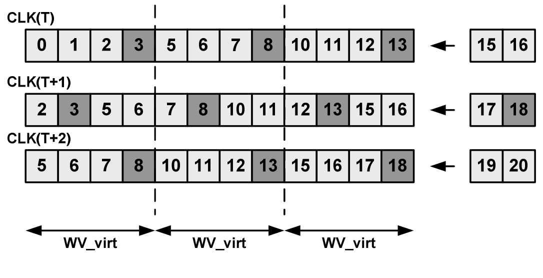


Figure 5. Masked data register

The output ranking part is the same as in the single-word case. The number of required equality comparators scales with the size of the real filter window, as it is sufficient to select the appropriate output when all samples in a new column have been inserted into the filter. In these cycles the locations of the valid samples are well defined.

D. Multi-word filter with multiple outputs

In case real samples are used for padding, the virtual filter kernel can be viewed as *NP+1* real filter windows joint together, where NP is the number of padding lines added to the filter window to form the virtual filter kernel. E.g. the 3x4 virtual kernel on Fig. 4 can be viewed as two 3x3 TAP filter windows joint together. The Filter Core presented in the previous section already computes all the required comparison results to generate valid outputs for both filter windows, however the mask generator, the one-counters and the output address generator should be replicated. The advantage is that the relation between the operating frequency and the number of new inputs processed in a single cycle becomes even better:

$$FO = \frac{\left\lceil \frac{WV}{NI} \right\rceil}{\left\lceil \frac{WV}{NI} \right\rceil * NI - WV + 1} * FS \qquad (11)$$

The drawback is that the Line Buffer should store *WV* lines of the input image instead of *WV-1*, and in case of real-time video filtering an output buffer is also required.

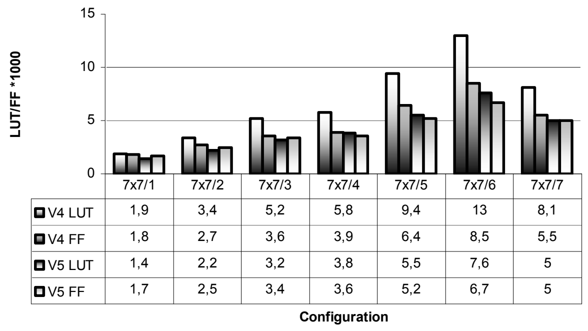
IV. Implementation results

The following implementation results were obtained using 24 bit RGB input, while the FVG was set to sum the three color components and output the 10-bit result. Table II summarizes the obtainable operating frequency of the word-serial architecture in different Xilinx FPGA families and different TAP numbers.

TABLE II. WORD-SERIAL OPERATING FRQUENCY

Family	**TAP number**		
	9	***25***	***49***
XC5V-3*	480 MHz	480 MHz	480 MHz
XC4V-10	400 MHz	400 MHz	350 MHz
XC2V-5	235 MHz	225 MHz	215 MHz
XC3S-4	200 MHz	185 MHz	185 MHz

As the most demanding commercial video format (HDTV 1920*1080p) has a pixel frequency of 75 MHz, the required filter architecture can be easily selected based on the above table. For example, a Virtex-4 device can perform real-time filtering on HDTV source using a 49 TAP filter by employing a multi-word Filter Core configuration with 2 input samples per clock cycle. Fig. 6 summarizes the resource requirements of a 49 TAP rank filter using different Filter Core configurations (configuration: WVxWH/NI). LUT and FF denote the number of LUTs and flip-flops in Virtex-4 and Virtex-5 devices, respectively. As can be seen on Fig. 6, there are multi-word configurations (such as 7x7/5, 7x7/6) which require more resources than the full-parallel architecture (7x7/7). The reason for this is that the virtual filter kernel becomes way larger than the real filter window due to the enormous number of padding samples.



	7x7/1	7x7/2	7x7/3	7x7/4	7x7/5	7x7/6	7x7/7
▨ V4 LUT	1,9	3,4	5,2	5,8	9,4	13	8,1
▨ V4 FF	1,8	2,7	3,6	3,9	6,4	8,5	5,5
▨ V5 LUT	1,4	2,2	3,2	3,8	5,5	7,6	5
▨ V5 FF	1,7	2,5	3,4	3,6	5,2	6,7	5

Figure 6. Resource requirements

Obviously, these configurations should not be used; however, as can be calculated from Table 2, these are not required even in the slowest FPGAs.

V. Conclusion

An efficient architecture for high performance two dimensional rank filter was presented. Rank order filters, especially median filters, are used extensively for removing non-Gaussian (salt and pepper) noise from images and video feeds. Targeting FPGA implementations for video applications, a parametrizable structure was proposed, which deliver efficient solutions custom tailored to different pixel clock rates, available resources, and operating speeds. Compared to previous 2D architectures, the size and complexity of the filter structure was considerably reduced by optimally balancing the number of new input samples entered into the core and the available operating frequency of the filter. The proposed solution is independent of input data type, as it offers great flexibility to either generate magnitude information corresponding to RGB data, or can take advantage of preexisting magnitude information if such data is already available. The presented architecture can be further generalized to use arbitrarily shaped filter kernel and to perform weighted filtering.

References

[1] R. Roncella, R. Saletti, P. Terreni, "70-MHz 2-mm CMOS Bit-Level Systolic Array Median Filter", IEEE Journal of Solid State Circuits, vol 28, 1993.

[2] B. K. Kar and D. K. Pradhan, ``A new algorithm for order statistic and sorting,'', IEEE Trans. Signal Processing vol 41, August 1993.

[3] C. Chakrabarti and L. Wang, "Novel Sorting Network-Based Architectures for Rank Order Filters", IEEE Trans. On VLSI Systems, vol. 2, December 1994.

[4] F. A. Suhaib, P. Y. K. Cheung, L. Wayne: "Novel FPGA-Based Implementation of Median and Weighted Median Filters for Image Processing", Field-Programmable Logic and Applications (FPL 2005), 2005.

[5] C. Chakrabarti, "High Sample Rate Array Architectures for Median Filters", IEEE Trans. on Signal Processing, vol. 42, March 1994.

*: Post-synthesis result. Synthesis was done using Synplicity Premier.