Hindawi Publishing Corporation
VLSI Design
Volume 2008, Article ID 753043, 6 pages
doi:10.1155/2008/753043

Research Article

High-Performance Timing-Driven Rank Filter

Péter Szántó,[1] Gábor Szedő,[2] and Béla Fehér[1]

[1] *Department of Measurement and Information Systems, Budapest University of Technology and Economics, Magyar tudósok krt. 2, 1117 Budapest, Hungary*
[2] *Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124, USA*

Correspondence should be addressed to Péter Szántó, szanto@mit.bme.hu

Received 27 April 2007; Accepted 2 November 2007

Recommended by Jean-Baptiste Begueret

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 tradeoffs between complexity and performance. By maximizing the operating frequency, the complexity of the filter structure can be considerably reduced compared to previous 2D architectures.

1. INTRODUCTION

Rank-order filtering is a nonlinear 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 is discarded and a set of new elements is 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 impulses 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 preprocessing for edge detection. This paper presents a hardware architecture that is tailored for high-performance color video processing, but it can be used in various applications such as IP block by taking advantage of design time parameterization. The paper concentrates on the timing-driven architecture selection which exploits the high operating frequency of recent FPGA and ASIC technologies, thus reducing hardware resource requirements.

2. 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 they do not lend themselves well to high sample rate implementations as filtering performance is proportional to the precision of the input data. However, the processing rate typically does not depend on the number of samples which changes between processing cycles.

Insert/delete or sorting network-based architectures [3, 4] explicitly order 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 stores the samples in the order of arrival and selects the appropriate output sample by calculating the location of the output sample dynamically. These architectures are easier to pipeline and they still require few comparators.

3. PROPOSED ARCHITECTURE

On filtering images or videos, the filter window is sliding horizontally across the input image, as illustrated in Figure 1. In

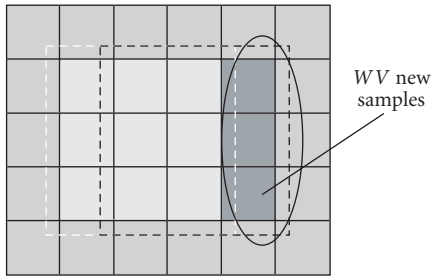

Figure 1: 2D image filtering.

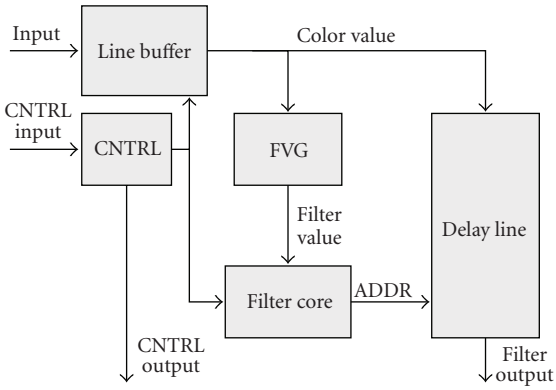

Figure 2: Global filter architecture.

case of a simple rectangular window, to generate a valid output, WV (vertical size of the filter window) new input samples should be processed. Word-serial architectures can process one input sample per clock cycle. When comparing different solutions, an important classification criterion is the level of input parallelization. In the 2D filtering case, the filter should operate at WV times of the input pixel frequency and generate a valid input sample every WVth clock cycle.

Fully parallel filters can generate a valid output sample every clock cycle, irrespective of the number of input samples required to achieve this process. Consequently, such filters process WV new samples in a single clock cycle, and the required operating frequency is equal to the input pixel frequency. At the same time, hardware resource requirements are greatly increased. Previous papers typically considered fully parallel architectures such as 2D filters; however, as this paper proves, using recent FPGA technologies, this solution is suboptimal due to the inefficient resource utilization.

Multiword architectures are hybrid solutions; in one cycle, they can handle more than one input sample, but less than the fully parallel implementation. This solution allows finding an optimal balance between operating frequency and hardware complexity. Using given filter window and input pixel frequency, with NI defining the number of new input samples in a single cycle, the required operating frequency can be computed as

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

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

3.1. Global filter architecture

The proposed architecture consists of five main components (as illustrated in Figure 2): the line buffer (LB), the optional filter value generator (FVG), the delay line (DL), the filter core (FC), and the control unit (CNTRL).

The LB stores WV-1 lines of the original input frame in the internal memory. The FVG is only required if the input format does not contain a magnitude-like component. For YCbCr or YUV input representations, this module can be omitted as the Y component lends itself well to magnitude ordering. For RGB input (luminance), a typical magnitude value can be calculated. The DL is an addressable FIFO which stores the full per-pixel information of the pixels residing inside the FC. The FC itself uses the values computed by the FVG and generates the appropriate address for the DL. CNTRL generates properly delayed synchronization signals and output valid signals. As the rest of the architecture is independent of the FC solution, further discussion will focus on the FC and its extensions.

3.2. Word-serial filter core

The operation of the FC 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 is equal 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 (the smallest sample) to TAP-1 (the largest sample). The ranked sample is the one which has the index value equal to the required rank. The block diagram in Figure 3 illustrates the hardware implementation of the algorithm for $TAP = 5$. 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 (the result is "1" if the new value is smaller than the older ones, and "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, the full content of the $CR[\,]$ registers is

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

where (:) denotes bit selection, {} denotes concatenation, and $C[k]$ denotes the kth comparison result. The comparison result of a given value is shifted to the left together with

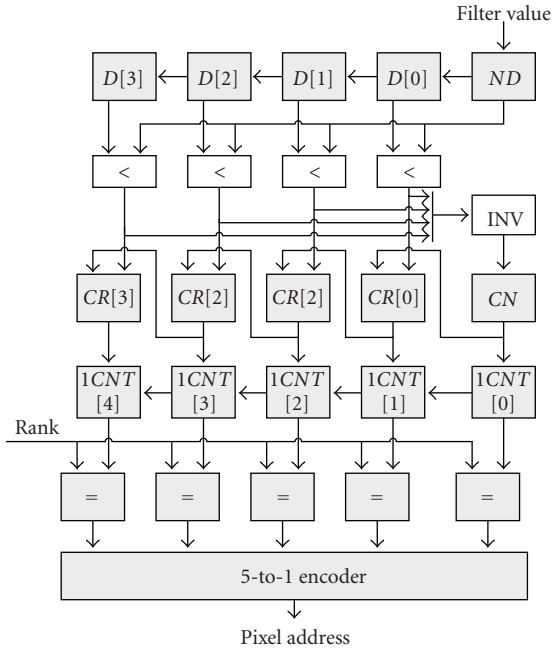

Figure 3: Filter core.

Table 1: Filtering example.

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

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 a number of values which are smaller than the given value. These bit summing operations are carried out 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 arbitrarily from clock to clock. Generation of $CR[k]$ can be optimized taking into consideration the fact that only two bits change 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 results of the bit summing blocks are compared with the required rank, generating a *TAP* bit wide vector of results containing exactly one "1" at the position of the cell which contains the required output. An encoder passes this position to the DL as an address. Table 1 shows an example with the data registers ($D[\,]$, ND), $CR[\,]$, CN and the output of the 1CNT blocks.

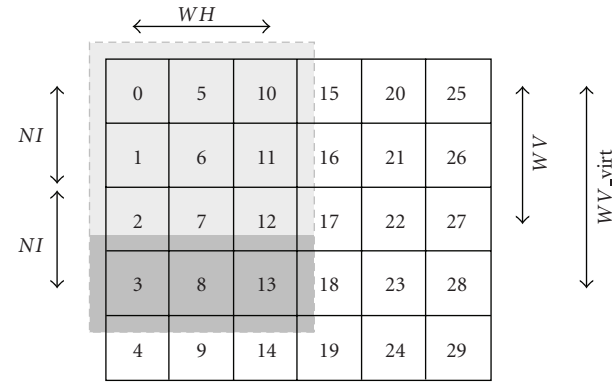

Figure 4: Virtual filter kernel.

3.3. Multiword filter core

The architecture presented in the previous section can be easily extended to process more than one new filter value per clock cycle. Instead of one, the data registers ($D[\,]$) and the comparator result shift registers ($CR[\,]$) should shift by NI data positions. 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 is

$$C = (TAP - NI)^{*}NI + \frac{NI^{*}(NI-1)}{2}. \tag{3}$$

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 always suffice as the number of valid new inputs is either NI or $WV \bmod NI$ (see Figure 4). Still, for large apertures, numerous multiplexers may be required.

Another solution is to insert padding samples as necessary such that in every clock cycle NI new samples can be entered, thus creating a virtual filter kernel (VK). Figure 4 illustrates such kernel for $WV = 3$ and $NI = 2$ 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 are irrelevant). Obviously, this method makes the size of the VK larger than that of the real filter window, hence requiring more hardware resources as parts of the FC scale with the size of the VK.

Figure 5 presents the contents of the data registers clock by clock, using the example in Figure 4, as new inputs are inserted and the filter window is moved horizontally. Background shading of valid and invalid (padding) samples corresponds to Figure 4. Samples on the right are the input samples. As any given register may contain valid or invalid samples during operation, comparisons are done using all data

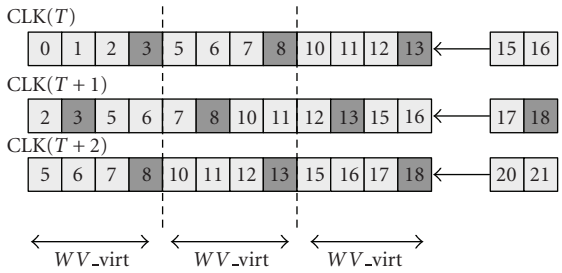

Figure 5: Masked data register.

registers, irrespective of the validity. Therefore, the number of comparators required scales with the size of the VK.

Padding samples are masked after the comparator result registers (*CR*[], *CN*[]), but before the 1CNT blocks. For each older sample, masking is done for 2**NI* bits. *NI* bits mask the comparison results with the *NI* new samples, and other *NI* bits mask the comparison results of the oldest *NI* samples. The output ranking part is the same as in the single-word case. The number of required equality comparators is proportional to 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.

3.4. Multiword filter with multiple outputs

In case valid samples are used for padding, the virtual filter kernel can be viewed as *NP + 1* filter windows processed together, where *NP* is the number of padding lines added to the filter window to form the VK. For example, the 3×4 virtual kernel in Figure 4 can be viewed as two 3×3 partially overlapping filter windows. The FC presented in the previous section already computes all the required comparison results to generate valid outputs for both of the 3×3 filter windows. However, to come up with 2 separate outputs, 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, significantly improving efficiency:

$$FO = \frac{\left\lceil \frac{WV}{NI} \right\rceil}{\left\lceil \frac{WV}{NI} \right\rceil * NI - WV + 1} * FS. \quad (4)$$

The drawback is that the LB should store *WV* lines of the input image instead of *WV−1*. In case of real-time video filtering, an output buffer may also be required.

3.5. Nonrectangular filter window

The mask-based filtering architecture allows for the easy implementation of nonrectangular (convex and nonconvex) filter windows. The most significant difference compared to the multiworld implementation described above is that the valid

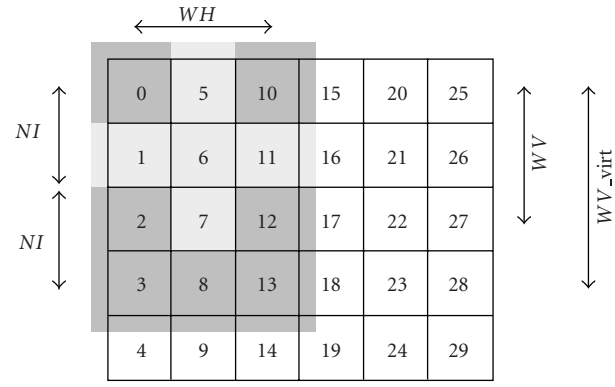

Figure 6: Nonrectangular mask.

or invalid status of a given filter value may change as the filter window slides across the input image. For example, in Figure 6, pixel 10 is invalid in the first computation cycle it is used, but as the window slides one pixel to the right, it becomes valid.

Consequently, bit summing becomes more complex as the number of possible transitions between the masked *CR*[] and *CN*[] registers is increased. Nonrectangular windows typically increase the number of invalid samples within the VK. Therefore, using the bit summing block for the valid samples only may reduce resource requirements. Practically, in the latter implementation, only the number of *ND*[] and *D*[] registers scales with the virtual filter window; all other processing units are implemented only for the valid data.

3.6. Weighted rank filtering

Some applications require the use of weighted filter windows, rendering some input samples more significant than others when determining the output of the filter. The proposed method allows for the application of integer weights. The comparison result bits (*CR*[] and *CN*[] registers' outputs) are replicated as many times as determined by the corresponding weight factor. However, the bit summing blocks become increasingly complex as their inputs become wider due to bit replication. Also, the *TAP* bit summing operation results in *TAP* different values, which are in the range of $0 \cdots W-1$, where *W* is the summation of all the weights. As *TAP* is smaller than *W*, not all integer values will be presented at the outputs of the bit summing units. Therefore, a simple equality comparator is no longer adequate to determine the ranked sample. Instead, the filter has to find the sample which has the closest bit summing value to the required rank (which is in the range of $0 \cdots W-1$).

To facilitate the correct selection, the proposed architecture (see Figure 7) employs several difference computing units and a selection tree. The difference computing units process the required rank and the outputs of the bit summing units. The two input minimum calculators select the smaller of their inputs together with a binary flag which shows whether the left or the right input was selected. At the root of the tree, the concatenated tag bits determine the

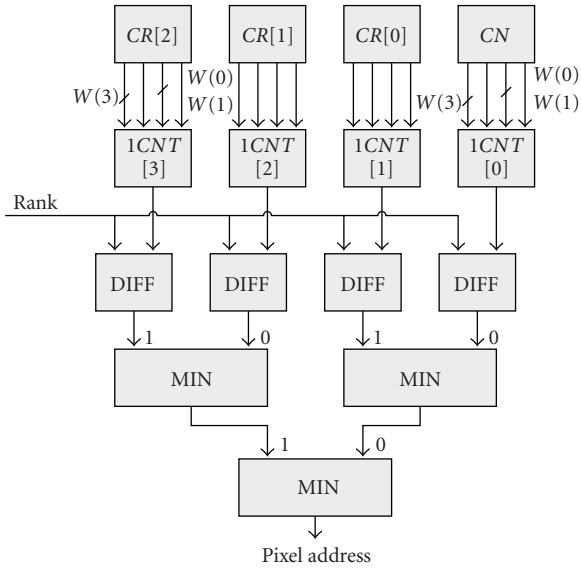

Figure 7: Filter core for weighted rank filtering.

Table 2: Word-serial operating frquency.

Family	Number of taps (*TAP*)		
	9	25	49
XC5V-3	460 MHz	420 MHz	400 MHz
XC4V-10	400 MHz	375 MHz	355 MHz
XC3S-4	245 MHz	195 MHz	175 MHz

location of the sample which has the closest bit summing value to the required rank. This value can be used to address the DL.

4. IMPLEMENTATION RESULTS

The following implementation results were obtained using 24-bit RGB input, with an FVG that sums the three color components and outputs a 10-bit result. Table 2 summarizes the operating frequencies obtained for the word-serial architecture for different Xilinx FPGA families and different *TAP* numbers. These values can be used as a reference to help determine the required parallelization level of the FC, depending on the input pixel frequency and the filter window size.

Table 2 offers different solutions even for one of the most demanding commercial video format, HDTV1080p, which has a pixel frequency of 75 MHz. For example, a Virtex-4 device can perform real-time filtering on HDTV source using a 49-tap filter by employing a multiword FC configuration with 2 input samples per clock cycle. Figure 8 summarizes the resource requirements of a 49-tap rank filter using different FC configurations (configuration *WVxWH/NI*). LUT and FF denote the number of lookup tables and flip flops in Virtex-4 and Virtex-5 devices, respectively. Figure 6 demonstrates that some multiword configurations (such as 7×7/5, 7×7/6) may require more resources than the full parallel architecture (7×7/7). The reason for this is that the VK becomes much larger than the valid filter window due to the enormous number of padding samples.

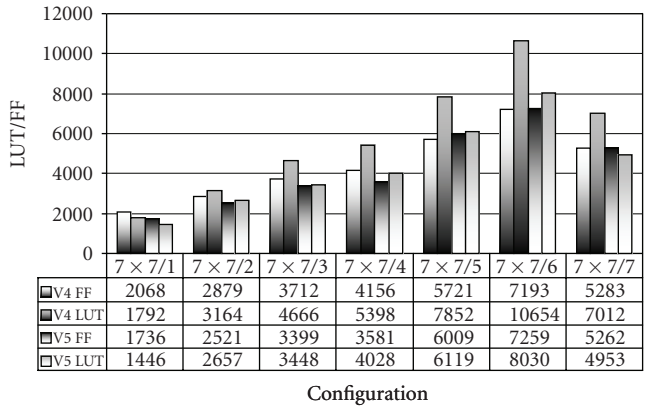

Figure 8: Resource requirements.

Table 3: Operating frequency and resource requirements using Spartan-3 (9 and 25 taps).

Configuration	3×3/1	5×5/1	5×5/2	5×5/3
FFs	567	1234	1832	2044
LUTs	395	937	1420	1913
BRAMs	6	12	12	12
$F_{CLK\,max}$	245	195	180	165

Table 4: Operating frequency and resource requirements using Spartan-3 (49 taps).

Configuration	7×7/1	7×7/2	7×7/3	7×7/4
FFs	2121	3030	3849	4242
LUTs	1862	3302	4866	5627
BRAMs	18	18	18	18
$F_{CLK\,max}$	175	160	150	150

These configurations are inferior to the full parallel architecture in terms of throughput and silicon real estate. The presented architecture can take advantage of the 6-input LUTs of the Virtex-5 FPGA family, resulting in 20–30% reduction in the design size.

Tables 3, 4, 5, and 6 summarize the achievable operating frequencies and resource requirements of several filter configurations using Spartan-3, Virtex-4, and Virtex-5 devices, respectively. For every filter size and FPGA family, the configurations marked with light grey background can be used to filter HDTV (1920×1080 30 p—75 MHz pixel clock) input. The lower-performance configurations are still adequate for lower-resolution video inputs, like SDTV.

Although the longest register-to-register path does not depend on the filter configuration, as the complexity of the filter increases, the achievable operating frequency still decreases. This is common when using FPGAs and should be taken into consideration when selecting the filter configuration for given input format and filter size.

Table 5: Operating frequency and resource requirements using Virtex-4.

Configuration	3×3/1	5×5/1	7×7/1	7×7/2
FFs	594	1088	2068	2879
LUTs	406	950	1792	3164
BRAMs	6	12	18	18
$F_{CLK\,max}$	400	375	355	300

Table 6: Operating frequency and resource requirements using Virtex-5.

Configuration	3×3/1	5×5/1	7×7/1	7×7/2
FFs	580	1050	1736	2521
LUTs	290	750	1446	2657
BRAMs	3	6	9	9
$F_{CLK\,max}$	460	420	400	340

5. CONCLUSION

An efficient architecture for high-performance two-dimensional rank filters was presented. Rank-order filters, especially median filters, are used extensively for removing non-Gaussian (salt and pepper) noise from images and video streams. Targeting FPGA implementations for video applications, a parameterizable structure was proposed which delivers an efficient solution 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 were considerably reduced by 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 generate magnitude information corresponding to RGB data, or it can take advantage of preexisting magnitude information if such data are already available. The solution presented can handle nonrectangular filter windows or weighted samples as well, which widens the domain of possible applications even further.

REFERENCES

[1] R. Roncella, R. Saletti, and P. Terreni, "70-MHz 2-μm CMOS bit-level systolic array median filter," *IEEE Journal of Solid State Circuits*, vol. 28, no. 5, pp. 530–536, 1993.

[2] B. K. Kar and D. K. Pradhan, "New algorithm for order statistic and sorting," *IEEE Transactions on Signal Processing*, vol. 41, no. 8, pp. 2688–2694, 1993.

[3] C. Chakrabarti and L.-Y. Wang, "Novel sorting network-based architectures for rank order filters," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 502–507, 1994.

[4] P. Szántó, G. Szedő, and B. Fehér, "Implementing 2D median filter in FPGAs," in *Proceedings of the 7th International Carpathian Control Conference (ICCC '06)*, Roznov, Czech Republic, May 2006.

[5] C. Chakrabarti, "High sample rate array architectures for median filters," *IEEE Transactions on Signal Processing*, vol. 42, no. 3, pp. 707–712, 1994.