

Cost-Effective Two-Dimensional Rank-Order Filters on FPGAs

Xilinx Virtex FPGAs provide excellent co-, pre-, and post-processing hardware acceleration solutions.

by Gabor Szedo
Senior DSP Engineer
Xilinx, Inc.
gabor.szedo@xilinx.com

Peter Szanto
Research Assistant
Budapest University of Technology and Economics,
Department of Measurement and Information Systems
szanto@mit.bme.hu

Rank-order filtering is a non-linear filtering technique in which an element is selected from an ordered list of samples. Two-dimensional (2D) filtering is performed on the contents of a rectangular window that slides across an image. As the window moves by one pixel, a set of obsolete elements are discarded and a set of new elements are inserted into the filter window. The samples within the window are sorted and the element with the specified rank is selected as output. Most typical ranks are median, minimum, and maximum.

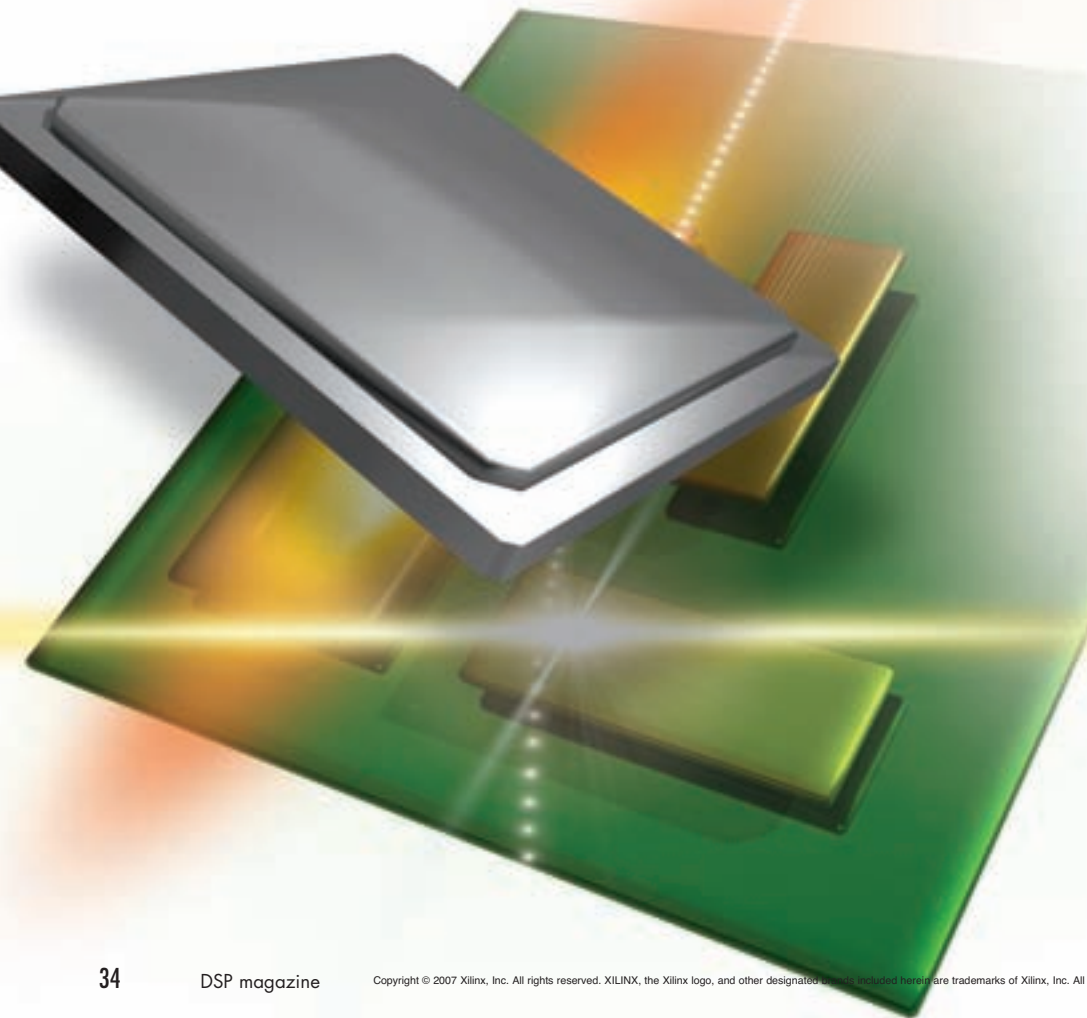
Compared to linear filters such as finite impulse response (FIR) or IIR, rank filters can effectively remove specks while preserving edges. This can be very useful for noise-removal or pre-processing applications. In this article, we'll present an architecture that lends itself well to area and performance trade-offs in high-performance FPGAs.

Previous Work

Earlier rank-filter implementations have not dealt with aspects of color image processing, predominantly classifying a filter as 2D if it was able to generate a valid pixel output in a single clock cycle.

Bit-serial approaches provide the lowest complexity. Processing rates are not usually dependent on the number of new samples the filter can handle in a single clock cycle. Bit-serial approaches are non-recursive and consequently easy to pipeline, but require a large number of comparators. Filtering performance is proportional to input data width.

Word-parallel architectures usually implement a sorting network that employs bubble sorting, odd/even merge sorting, and other architectures optimized for resource efficiency.



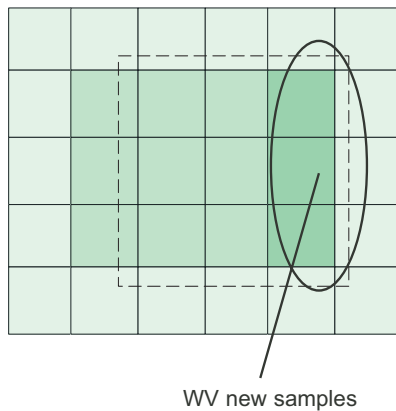


Figure 1 - 2D filtering window

Most rank-filter architectures either store samples by the order of arrival (FIFO) or magnitude ordered. Insert/delete architectures store samples ordered by magnitude. The oldest sample is discarded and the most recent input is inserted into the sorting structure at the appropriate location. These solutions require fewer comparators.

FIFO-based architectures dynamically calculate the location of the output sample. These architectures are easier to pipeline and adapt to multiple samples per clock cycle, which is essential for 2D filtering.

The Xilinx 2D Rank-Filter Architecture

Other than larger tap numbers, the most important difference between 2D and 1D rank filtering is that multiple input samples must be inserted into the 2D filter core to generate one new output sample. Figure 1 shows the pixel high-sliding filter window, which has a vertical size variable we'll call WV, moving across the input image from left to right. The smaller gray squares represent pixels while darker squares illustrate new samples.

You can trivially extend 1D filters for 2D use by operating the filter at a WV multiple of the pixel clock, reading new input pixels every clock cycle but generating valid output pixels only once every WV clock cycle. Depending on the filter size and the targeted FPGA technology, this solution is viable for a wide range of applications. You can filter windows as high as 3 pixels on most FPGAs and as high as 5 pixels on Xilinx®

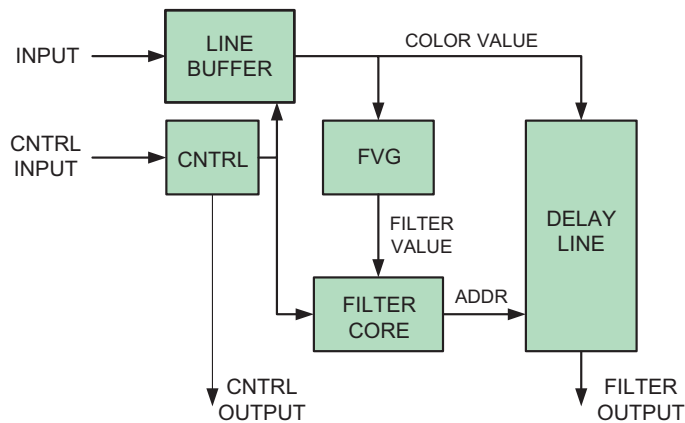


Figure 2 - Filter architecture

Virtex™-4 devices or Virtex-5 devices using pure 1D architectures for 75-MHz 1080p HDTV applications.

If pixel clock frequencies are prohibitively high, parallel instances of certain key components of the filter may accept new WV input samples every clock cycle.

Hybrid solutions spanning fully parallel (WV input samples per clock cycle) and word serial (one input sample per clock cycle) allow you to tune the filter core to the maximum clock frequency allowed by the target chip while minimizing resource counts.

Basic Filter Architecture

The architecture illustrated in Figure 2 comprises five main components. The line buffer stores WV-1 lines of the input frame. If required, the filter value generator (FVG) computes magnitude values (such as luminance) for filtering. The delay line block stores full pixel information (such as RGB) for the pixels currently being processed by the filter core, which carries out the actual rank filtering. The control block generates optional data switching, masking, and output valid signals.

For typical resolutions and filter sizes, you can implement the line buffer using block RAMs internal to the target FPGA.

Color frames are often filtered using a function of the RGB values – typically luminance – rather than the full RGB information. Color information (corresponding to the luminance values

processed by the filter core) is stored in a FIFO. Let's designate the variable TAP to mean the size of the filter core; at any given time the number of pixels the FIFO stores is the sum of TAP and the latency of the filter core. The SRL16 primitives in Xilinx FPGAs offer an efficient way to implement this addressable FIFO.

Filter Core

Each sample in the filter core is coupled with an index value, which represents the number of samples less than the corresponding sample. When a new sample is inserted into the window, the samples already in the filter are compared to the new sample. Based on these comparisons the index values are updated, resulting in TAP-distinct values ranging from 0 (lowest sample) to TAP-1 (highest sample). However, this assumes that the values already in the filter core were unique. As new samples enter the filter, samples already in the filter are shifted along with their corresponding index values.

The architecture in Figure 3 illustrates the algorithm for a TAP equal to 5. There are five registers for storing data values (D[3 ... 0]) and the new data sample (ND). Every sample is compared with a new sample to determine whether corresponding values are less than the new sample. The result of these comparisons is fed to TAP bit-wide shift registers (SH[4 ... 1]). Other bits of the SH registers are updated using comparison values calculated in previous cycles.

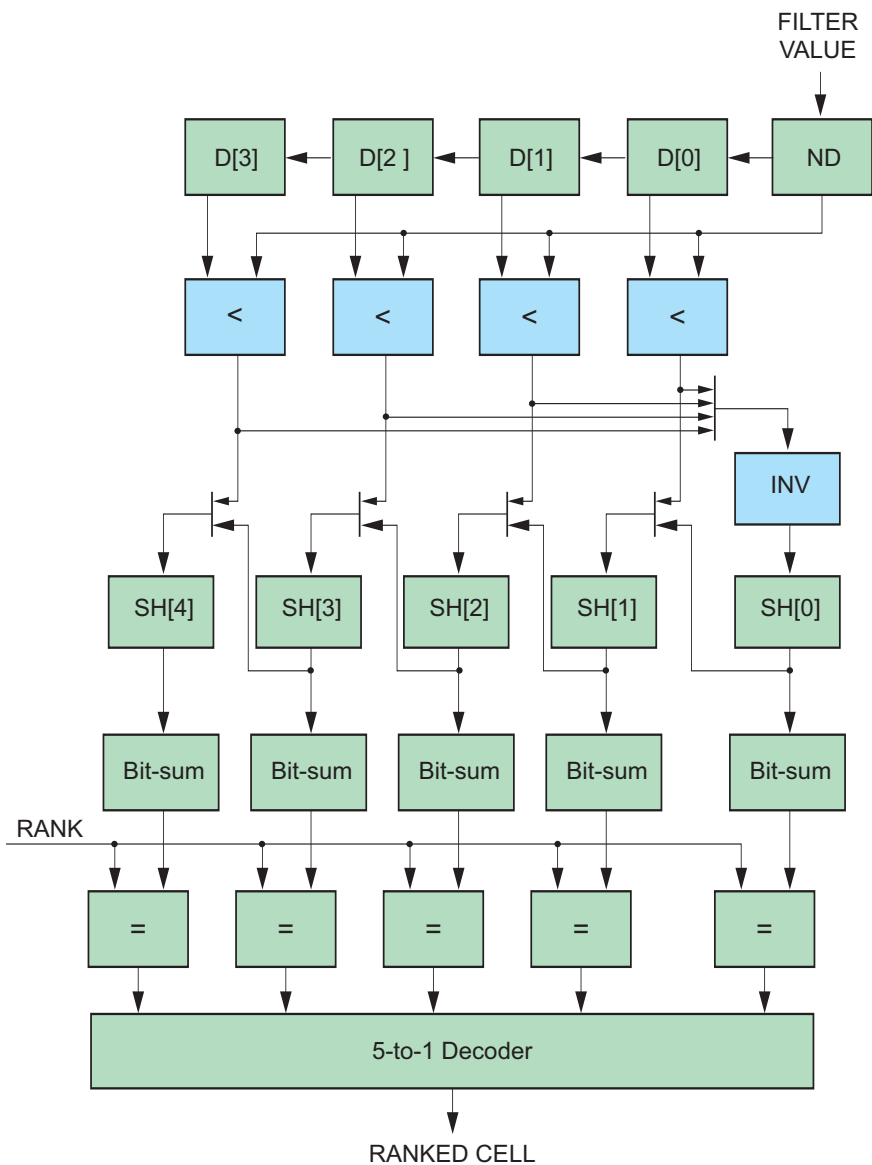


Figure 3 – Filter core architecture

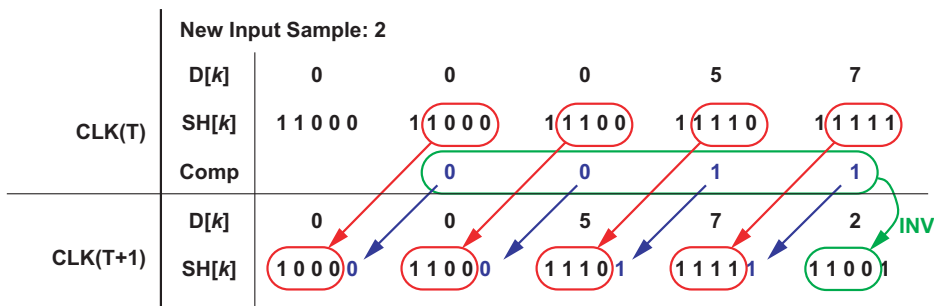


Figure 4 – Filter core operation example

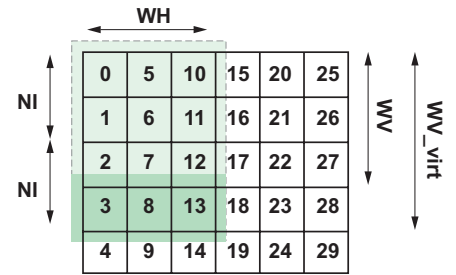


Figure 5 – Virtual filter window

At any given time, shift registers SH[k] store the comparison results of the corresponding sample D[k], with other samples residing in the filter. The sum of bits in the SH registers generates the index information.

Bit b (b>0) in SH[0] is updated with the negated comparison result of the comparator b. Bit 0 of SH[0], the self-comparison result, is initialized with a 1. Figure 4 illustrates the algorithm by presenting two cycles of the 5-TAP example as new samples enter from the right. Table cells show the contents of the data register (ND, D) and the corresponding shift register (SH) values.

A straightforward way to sum up the number of 1s in the shift registers is to use adder trees. For the new sample, this is the only solution, as contents of SH[0] can change arbitrarily between subsequent clock cycles. To calculate the sum of SH[TAP-1 ... 1] bits, you can use an increment/decrement structure, taking advantage of the correlation between subsequent SH values. For small TAP numbers, adder trees are less complex. As the TAP number increases, the latter solution becomes more efficient.

The performance of the 1D filter structure plays a key role in evaluating 2D implementation options. Table 1 summarizes the maximum operating frequency for different FPGAs and different TAP numbers.

2D Extension

You can easily extend the 1D architecture to process more than one new sample per clock cycle. The data registers and the comparator result shift registers should shift by NI data positions, where NI represents the number

of new input samples per clock cycle. The number of comparators is proportional to NI, as old samples should be compared with all new samples and new samples should be compared with each other.

If WV is not an integer multiply of NI, the throughput of the filter core input exceeds that of the input stream, so in some clock cycles the number of valid new data samples will be less than NI. By inserting multiplexers in front of data and comparator result shift registers, the filter can process a dynamically changing number of new samples. Even though two to one multiplexers are always sufficient, numerous multiplexers

are required to facilitate dynamic shifting.

Another solution is to insert padding samples as necessary so that new NI samples can be entered in every clock cycle. The actual filter uses a WV_{virt} high virtual filter window. Figure 5 illustrates a virtual filter window for a case in which $WV = 3$ and $NI = 2$. Valid samples in the window are marked with light gray while padding samples are marked with dark gray.

During operation, all registers may contain both valid and padding samples. Comparisons are performed using all data registers irrespective of whether the actual sample is valid or not, so the number of

comparators required scales with the size of the virtual filter window. Padding samples are masked in the shift registers before counting comparator results. Masking values change according to the validity of new samples. For new samples, masking is performed on the entire TAP-wide registers, as contents can change from clock to clock. Mask value is periodic with WV and can be generated by a shift register.

Apart from bit masking, bit sum calculators for the new samples are the same as for the 1D case. Counters for the older samples get more and more complex as NI increases because the modification value is extended from the $(-1, 0, +1)$ range to $(-NI \dots +NI)$.

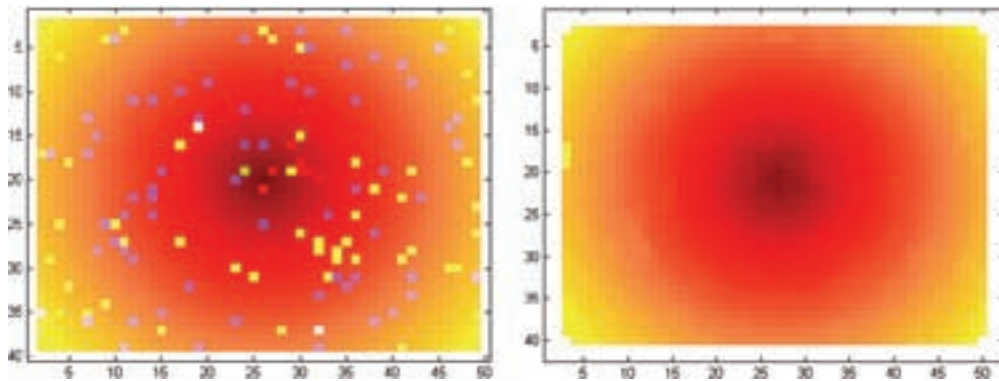


Figure 6 – Salt and pepper noise removal using a 3 x 3 median filtering window

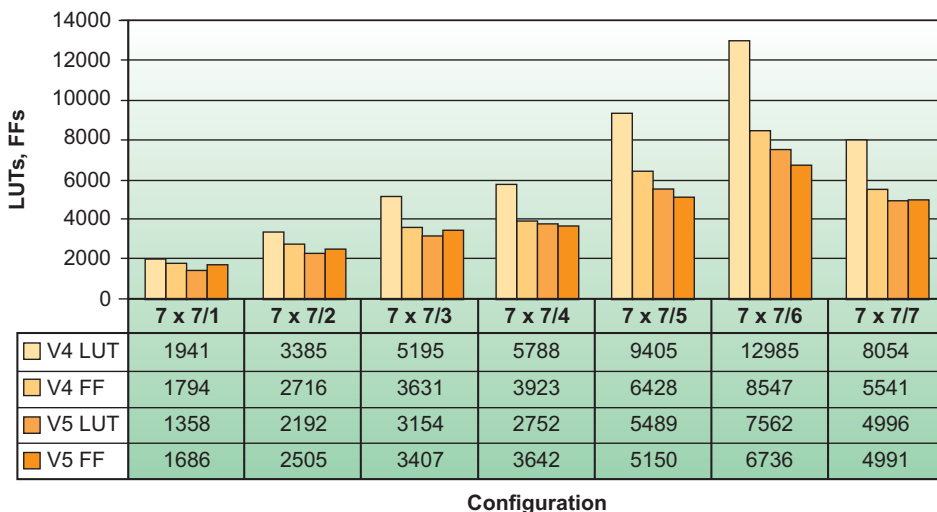


Figure 7 – LUT and flip-flop usage as a function of NI for a 7 x 7 window size

Conclusion

Two-dimensional rank order filtering is a non-linear filtering technique that can effectively remove “salt and pepper” noise from images or video streams (Figure 6). We implemented a scalable, efficient 2D filter design on Spartan™-3 devices, Virtex-4 devices, and Virtex-5 devices.

Table 1 summarizes the achieved operating frequencies of the 1D filter core for typical window sizes. For characterization purposes, comparators were implemented using logic-fabric adders. Figure 7 presents LUT usage as a function of new input samples/CLK cycle for a 7 x 7 window size.

For more information, see Xilinx Application Note XAPP953, “Two-Dimensional Rank Order Filter,” at www.xilinx.com/bvdocs/appnotes/xapp953.pdf.

Family	Filter Window Size		
	3 x 3	5 x 5	7 x 7
XC5V-3	460 MHz	420 MHz	400 MHz
XC4V-10	400 MHz	375 MHz	355 MHz
XC3S-4	245 MHz	195 MHz	175 MHz

Table 1 - Filter core operating frequencies for different filter sizes and families