International Carpathian Control Conference ICCC' 2006
Rožnov pod Radhoštěm, CZECH REPUBLIC
May 29-31, 2006

Implementing 2D Median Filter in FPGAs

Péter SZÁNTÓ[1] , Gábor SZEDŐ[2], Béla FEHÉR[1]
[1] Department of Measurement and Information Systems,
Budapest University of Technology and Economics,
Budapest, Hungary, {szanto, feher}@mit.bme.hu
[2] Xilinx Inc. 2100. Logic Dr, San Jose, CA 95124, USA, gabor.szedo@xilinx.com

Abstract: This paper presents an FPGA implementation of a two-dimensional median filter architecture for image and video processing applications. The architecture exploits sorting based on partial rather than complete per-pixel information. This allows performance enhancement, which is a key point in image filtering, as the sampling frequency is typically quite high.

Key words: FPGA, image processing, filtering, median.

1 Introduction

Unlike FIR or IIR filters, the median filter is a non-linear algorithm with interesting properties: it can effectively remove impulse like noises, while preserving the edges of the input signal.

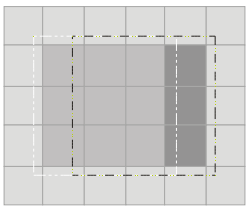

Figure 1. 2D Median filtering

There are two basic types of median filters: non-recursive and recursive. The former works by sorting the input samples in the filter window, and outputting the median of the sorted values. In 2D case a W_H*W_V window is moved across the input image (or frame), where W_H denotes the horizontal and W_V denotes the vertical size of the window in pixels. The center sample of the window is replaced by the median of the samples within the window (Figure 1). The current window is marked with white box, the next window is marked with black window, new samples are dark grey. The recursive method uses input samples, as well as previous output samples for filtering, thus allowing faster response to the changing properties of the input. This paper mainly focuses on non-recursive filters, but the presented architecture is also applicable for recursive filtering.

2 Previous work

There are lots of different architectures for implementing median filters both in software and in hardware. Software implementations vary from simple full-sorting to more special algorithms, which take advantage of special requirements,s such as the fact that only a fraction of the data changes from time to time.

Hardware implementations can be classified by different properties. Bit-serial approaches ([1]) do not explicitly sort the input data, instead select the appropriate sample as the output by inspecting the bits of the input samples in successive clock cycles. Thus, performance is proportional with the input data width (but not necessarily with the filter window size); for high

resolution these architectures may limit the sample or pixel rate. Word-parallel architectures can either store the samples in the order of arrival [2] or in sorted order. Sorted order architectures can be implemented as sorting networks ([3]) or systolic arrays. The architecture presented in this paper belongs to the second group – while sorting networks can be easily pipelined, they require a large number of comparators. On the contrary, the presented method requires minimal number of comparators.

3 Filtering in 2D

Figure 1. illustrates that moving the filter window with one pixel column requires W_V new samples to be entered, and W_V old samples to be discarded from the filter kernel. 2D filters can be partitioned into two groups based on the method they handle these new samples.

Architectures originating from 1D filters can handle one new input sample in one clock cycle. Therefore, they require W_V clock cycles to generate one valid output, so for real-time processing the filter operating frequency should be the pixel clock multiplied with W_V.

Another set of architectures process W_V number of samples in a single clock cycle. Although operating frequencies of these filters are much lower (equal to the pixel frequency), hardware resource requirements are larger.

Mixed mode architectures can process two or more input samples in one clock cycle, thus reducing operating frequency requirements at reasonable complexity.

4 Proposed Architecture

The proposed architecture is part of a video processing system, which receives standard, de-interlaced PAL or NTSC signal. The input resolution is 720x576 pixels, at 25 frames/second. Thus, the raw pixel clock equals to 10.368 MHz. Architectural decisions were based on these criteria and capabilities of modern FPGAs. The maximal required filter window size is 11x11 pixels, the architecture is a systolic array which can process one input sample in every clock cycle.

The top-level block diagram of the filter architecture is shown on Figure 2.

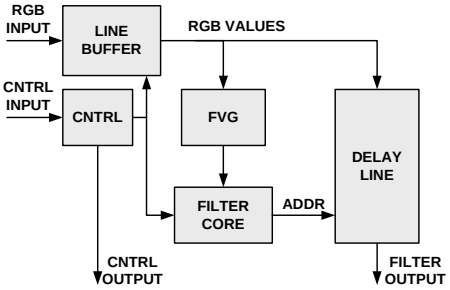

Figure 2. General architecture

The filter receives 24 bit RGB values and control signals (line end, frame end). The Input Front-end consists of a line buffer which stores W_V-1 number of lines and a Filter Value Generator, which produces an appropriate value for the sorting process.

The filter core itself generates the median output and the appropriately delayed versions of the control signals. The output of the Filter Core addresses an RGB delay line (which stores the RGB values of the filtering window) to generate the final output.

4.1 Input Front-end

The input front-end feeds the filter with pixels from $W_V \leq 10$ lines. Therefore, at most 10 input lines should be retained for further processing – they are stored in the line buffer. The required size of this buffer is

$$720*10*3=21600 \quad (1)$$

bytes, which can be stored in 11 BlockRAMs within the FPGA.

The output of the line buffers feed the Filter Value Generator. In the current implementation this unit simply sums the R, G and B components to form a luminosity-like value

$$Y= R+G+B. \quad (2)$$

Other, more complex operations, such as real RGB to YC_bC_r color-space conversion, can be also implemented as long as the conversion module can be easily pipelined. Y values retained for sorting can be represented on 10

bits. Note, that Y values can be similar for very different RGB values. As the sorting module can not take into account, this may distort the filtered image.

The original output of the front-end is also passed to the Filter Core, which provides the final output.

4.2 Filter Core and Delay Line

The Filter Core consists of three distinct parts: the filter cells, the empty generation and the delay line, as shown on Figure 3.

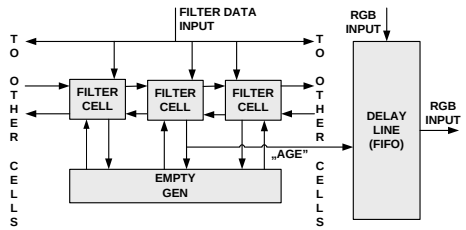

Figure 3. Filter core

A straightforward implementation may link the filter value and the full RGB value together, which is a trivial way to handle these values. However, as it will be obvious later, this would unnecessarily complicate the filter cells. Therefore, our architecture employs a delay line, which is an addressable FIFO storing RGB pixel values. The filter cells themselves do not generate color output but address this FIFO for RGB information. The address is the "age" of the sample on the output of the filter core. In Xilinx FPGAs, SRL16 primitive (16x1 bit, addressable shift register) lend themselves well for implementing delay lines.

The part responsible for the sorting is comprised of W_H*W_V similar cells. Each cell stores one sample of the filter window together with the "age" of the given sample. Age values show the number of clock cycles any given sample have been staying in the filter. Obviously, the sample with

$$age = W_H*W_V-1 \quad (3)$$

is the oldest, which it should be discarded.

In every clock cycle, each cell selects the appropriate function to either:

- Load a new sample
- Load sample from the cell to the right
- Load sample from the cell to the left
- Keep the current sample

In order to make a selection, cells have to know the results of comparisons between the new sample and sample values held by the cell itself and its nearest neighbors, as well as the position of the oldest sample.

To implement the above functionality, each cell consists of a comparator, an age counter and an "empty" register. The comparator compares the data of the cell with the new sample, the age counter counts the number of clock cycles, while the empty register shows if the sample of the cell is the one to be thrown away. Figure 4. shows the block diagram of a cell.

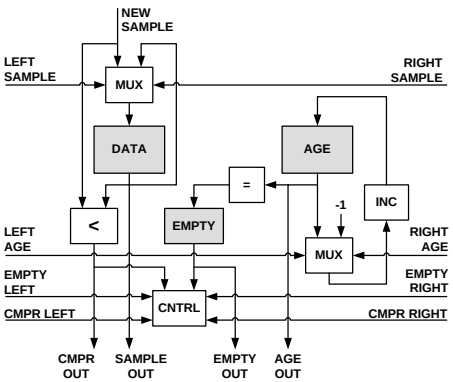

Figure 4. Cell block diagram

Darker blocks represent registers, while white blocks are combinatorial logic. DATA stores the filter data of the given pixel, LEFT_SAMPLE, RIGHT_SAMPLE and NEW_SAMPLE are the data of the left, the right adjacent cell and the new sample, respectively. Age variables are labeled with the same logic (note: the "-1" multiplexer input becomes "0" after the increment block). The CNTRL block generates all control signals for the multiplexers (connection not show on Figure 4.). The decision algorithm for a given cell can be summarized by the following pseudo code, where empty denotes if the current cell stores

the oldest sample, empty_right and empty_left signals the same for the right and left cells. Variables cmpr_curr, cmpr_left and cmpr_right hold the result of the comparison with the new sample for the current, the left and right adjacent cells, respectively.

```
if (empty)
    if (cmpr_left AND not cmpr_right)
        load new sample
    else if (not cmpr_left)
        load left sample
    else
        load right sample
else if (empty_right)
    if (not cmpr_curr)
        if (cmpr_left)
            load new data
        else
            load left data
    else
        keep current sample
else if (empty_left)
    if (cmpr_curr)
        if (not cmpr_right)
            load new sample
        else
            load right sample
    else
        keep current sample
```

Depending on the selected operation, the age counter may also load values from adjacent cells. When new sample is loaded, the counter is reset, when the right or the left sample is loaded, the corresponding incremented counter value is loaded, and when the current sample is kept, the counter is incremented.

For every cell, the empty_left and empty_right signals are generated asynchronously by an OR gate; empty_left is the result of combining all the empty signals of cells on the left of the given cell, whereas empty_right is the same for cells on the right of the given cell.

The output of the cell array is simply the age counter of the median cell. As the input of the filter cells has additional latency inserted by the computation of the filter value, the FIFO should be addressed by the latency compensated age counter value.

5 Improvements

As cells themselves have low latency, using this architecture for recursive filtering is quite straightforward. Cells use two 2:1 multiplexers; one at the filter value input and one at the input of the delay line. The multiplexer at the filter value input are fed by the input filter value and the filter value of the median cell, while the one at the delay line is fed by the input RGB and the output of the delay line.

Making the architecture capable of processing more than one input sample in one clock cycle is more difficult. Managing multiple empty cells and new samples makes the decision logic more complicated, while data multiplexers become more complex as well. This considerably reduces operating frequency; therefore for such applications a different architecture may be better suited.

6 Conclusion

Implementation results show that the presented architecture can reach 140 MHz in Virtex2-4 FPGAs and 250 MHz in Virtex4-12 FPGAs. Even the lower performance is more than sufficient to filter standard resolution PAL or NTSC video signals even with very large filter windows. Higher resolution videos are not typical in embedded systems, therefore the presented architecture completely satisfy the preliminary requirements. The small footprint of the design enables implementation using low-cost, small FPGAs.

References

LEE, C. L., JEN, C. 1993. *Binary Partition Algorithms and VLSI Architectures for Median and Rank Order Filtering*, IEEE Transactions on signal processing, Vol. 41, No. 9.

CHAKRABARTI, C., 1994. *High Sample Rate Array Architectures for Median Filters*, IEEE Transactions on signal processing, Vol. 42, No. 3.

CHAKRABARTI, C., WANG, L. 1994. *Novel Sorting Network-Based Architectures for Rank Order Filters*, IEEE Transactions on VLSI, Vol. 2, No. 4.