Sborník vědeckých prací Vysoké školy báňské - Technické univerzity Ostrava
číslo 2, rok 2006, ročník LII, řada strojní
článek č. 1554

Péter SZÁNTÓ[*], Gábor SZEDŐ[**], Béla FEHÉR[***]

IMPLEMENTING 2D MEDIAN FILTER IN FPGAS

IMPLEMENTACE 2D STŘEDOVÉHO FILTRU V FPGAS

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 relatively high.

Abstrakt

Příspěvek se zabývá FPGA implementací architektury 2D středového filtru v obrazových a video aplikacích. Architektura těží z výhodnosti rozdělení na části dat před skládáním kompletních per-pixel dat. Tento způsob umožňuje vylepšení, které je klíčové při obrazové filtraci, kde je vzorkovací frekvence relativně vysoká.

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.

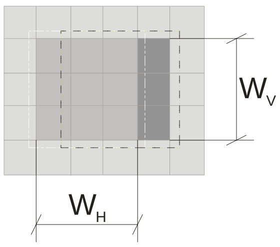

Fig. 1 2D Median Filtering

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 centre sample of the window is replaced by the median of the samples within the window (Figure 1). The current window is marked with white dashed square, the next window is marked with black dashed square; the new samples (which have to be processed to generate a new output) are dark grey. The recursive method uses input

[*] Péter Szántó, Department of Measurement and Information Systems, Budapest University of Technology and Economics, Magyar tudósok krt. 2., Budapest, (+36) 1-463-2066, szanto@mit.bme.hu
[**] Gábor Szedő, Xilinx Inc, 2100 Logic Dr., San Jose, (+1) 59 732 4380, gabor.szedo@xilinx.com
[***] Béla Fehér PhD, Department of Measurement and Information Systems, Budapest University of Technology and Economics, Magyar tudósok krt. 2., Budapest, (+36) 1-463-2686, feher@mit.bme.hu

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, 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 they 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 [4].

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 three 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 higher.

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 progressive 720x576 pixel sized video at 25 frames/second, thus the raw pixel clock equals to 10.4 MHz. Architectural decisions were based on these criteria and capabilities of modern FPGAs. The maximal required filter window size is 7x7 pixels, as this filter size is able to remove large noise pulses, while does not decrease the image too much. 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.

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, for example the luminance, or something close to this property.

The Filter Core itself generates a pointer to the median value, which addresses an RGB Delay Line (which stores the RGB values of the filtering window) to generate the final output.

The Control block calculates the address for the Line Buffer, generates the global control signals for the Filter Core and the appropriate output control signals, like line end and frame end.

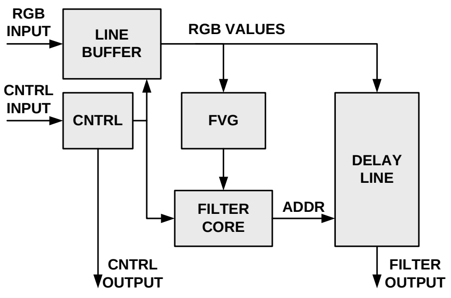

Fig. 2 General architecture

4.1 Input Front-end

The Input Front-end feeds the filter with pixels from $W_V=7$ lines in case of a 7x7 filter, therefore, 6 input lines should be stored internally for further processing – they are stored in the line buffer. The required size of this buffer is

$$720*6*3=12960 \qquad (1)$$

bytes, which can be stored in 7 BlockRAMs within the FPGA. As the Filter Core processes one input in a clock cycle, the Line Buffer is implemented in a 1D memory array storing the required number of input lines (in 7x7 case 6 lines are stored in a single memory array).

The output of the line buffers feeds the Filter Value Generator. Instead of computing the real luminosity value, the current implementation simply sums the R, G and B components to form a luminosity-like value

$$Y=R+G+B \qquad (2)$$

However, as long as the Filter Value Generator is pipelined, there is no limit on its complexity; therefore more complex calculations, such as real RGB to YCbCr color-space conversion, can be also implemented. 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 this into account, the filtered image may be distorted.

4.2 Filter Core

The Filter Core is responsible for the actual sorting process. It consists of W_H*W_V similar cells and an Empty Generator block. The middle cell connects to the Delay Line to address it.

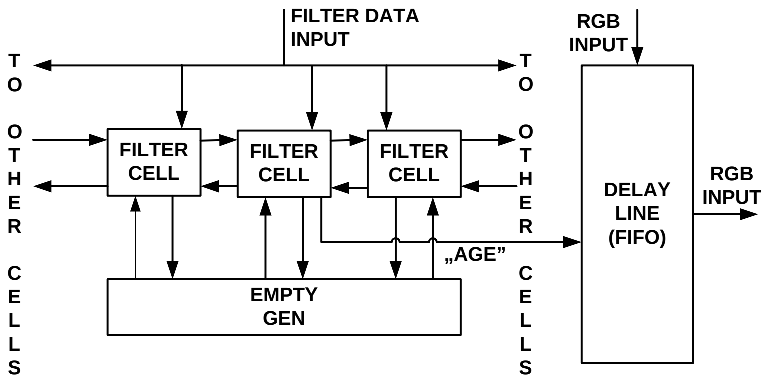

Fig. 3 Filter Core architecture

Each cell stores one sample (the filter value) 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

$$age=W_H*W_V-1 \tag{3}$$

is the oldest, which 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 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 neighbours, as well as the position of the oldest sample. The decision algorithm for a given cell can be summarized by the following pseudo code. Empty denotes if the current cell stores the oldest sample, empty_right and empty_left signals shows whether the oldest sample is on the right or the left of the current cell. 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 sample
      else
         load left sample
      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
```

To implement the above functionality, each cell consists of a comparator, an age counter, an "empty" signal generator and the decision (control) logic. The comparator compares the data of the cell with the new sample; the age counter counts the number of clock cycles the given data have been staying in the filter; while the empty signal shows if the sample of the cell is the one to be thrown away. Figure 4. shows the block diagram of a cell.

Grey blocks represent registers, while white blocks are combinatorial logic. DATA stores the Filter Value of the given pixel, LEFT_SAMPLE, RIGHT_SAMPLE and NEW_SAMPLE are the data of the left and the right adjacent cells and the new sample, respectively. Age variables are labelled with the same logic. The CNTRL block generates all control signals for the multiplexers (to make the diagram reasonably simple, control connections are not shown on Figure 4.).

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

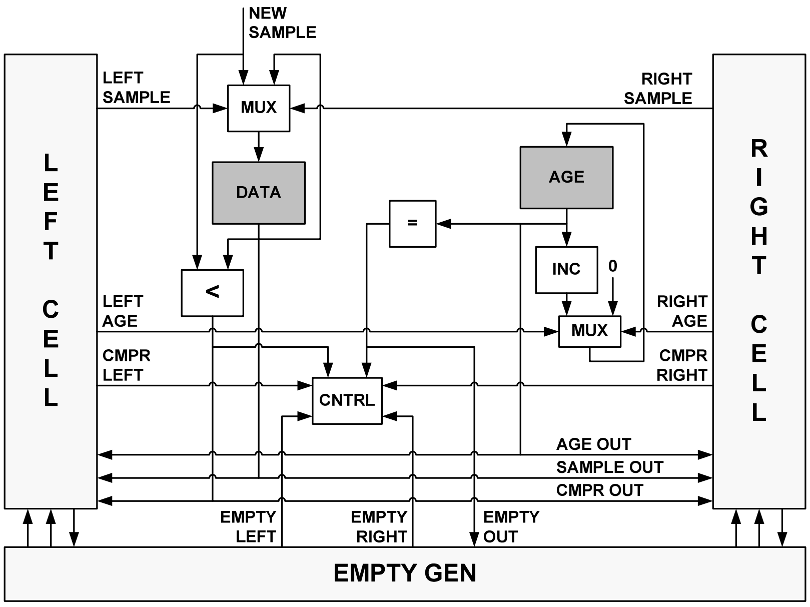

Fig. 4 Cell block diagram

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.

4.2. Delay Line

A straightforward implementation may link the filter value and the full RGB value together, which is a trivial way to handle these values. However, this method unnecessarily complicates the filter cells as there should be a register and a multiplexer for the RGB data path. Therefore, our architecture employs a delay line, which is an addressable FIFO storing the RGB pixel values. The filter cells themselves do not generate colour 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, output addressable shift register) lend themselves well for implementing delay lines.

5 RESULTS AND IMPROVEMENTS

As the performance and resource requirements are mainly defined by the Filter Core, Table 1. summarizes these parameters of this given sub-module for different Xilinx families, using a 9 TAP, a 25 TAP and a 49 TAP filter configuration. The maximal filter window size – assuming the previously derived video size – is 7x7 in Spartan-3, 9x9 in Virtex-4 and 11x11 in Virtex-5 FPGAs. During implementation, synthesis was done with Synplicity Premier 8.6, while mapping and place-and-route were performed by Xilinx ISE 8.2.

As cells themselves have low latency, using the architecture for recursive filtering is quite straightforward. In this case two additional 2:1 multiplexers are required; one at the filter value input of the cells and one at the input of the delay line The multiplexer at the filter value input are fed by the Filter Value Generator and the filter value of the median cell, while the one at the delay line is fed by the output of the Line Buffer and the output RGB of the Delay Line.

Tab. 1 FPGA resource requirements

TAP	FPGA	LUTs	FFs	Clock frequency
3X3 9	Virtex-5	397	179	250 MHz
	Virtex-4	606	146	180 MHz
	Spartan-3	629	147	100 MHz
5x5 25	Virtex-5	1160	487	200 MHz
	Virtex-4	1764	407	140 MHz
	Spartan-3	1759	402	95 MHz
7x7 49	Virtex-5	2060	845	170 MHz
	Virtex-4	3374	805	125 MHz
	Spartan-3	3477	833	75 MHz

6 CONCLUSIONS

Implementation results show that in the newest FPGAs real-time processing of standard resolution video signals can be achieved using relatively large filter kernels, while in cost-effective FPGA families the kernel size is more limited; however a 7x7 TAP filter is still possible. Higher resolution videos are not typical in embedded systems, therefore the presented architecture completely satisfy the preliminary requirements. The relatively small footprint of the design enables implementation using low-cost, small FPGAs.

REFERENCES

[1] 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.

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

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

[4] FEHÉR, B., SZEDŐ, G. *Noise Filter Applications on Reconfigurable Hardware*, PQACT '98 International Conference on Parallel Architectures and Compilation Techniques, Workshop on Reconfigurable Computing, Paris, France, 13th October 1998., pp 95-100

Reviewer: prof. Ing. Jiří Tůma, CSc., VŠB – Technical University of Ostrava