The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

A POSITIONAL FILTER SYNTHESIS FOR FPGA IMPLEMENTATION

Dariusz CABAN

Institute of Engineering Cybernetics, Wrocław University of Technology,
Janiszewskiego 11-17, 50-370 Wrocław, POLAND, *darek@ict.pwr.wroc.pl*

Abstract. *The paper reports on some experiments with implementing positional digital image filters using field programmable devices. It demonstrates that a single field programmable device may be used to build such a filter. By using extensive pipelining in the design, the filter can achieve performance of 50 million pixels per second (using Xilinx XC4000E devices) and almost 90 MHz (in case of Virtex-2 devices. These results were obtained using automatic synthesis from VHDL descriptions, avoiding any direct manipulation in the design.*

Key words: *Positional Filter, Median Filter, Synthesis, FPGA Implementation*

1. INTRODUCTION

The paper reports on the implementation of a class of filters used in image processing. The filtering is realised on a running, fixed size window of pixel values. Positional filtering is obtained by arranging the values in an ordered sequence (according to their magnitude) and choosing one that is at a certain position (first, middle, last or any other). Thus, the class of filters encompasses median, max and min filtering, depending on the choice of this position.

There are various algorithms used in positional filtering [7,10]. These are roughly classified to three groups: compare-and-multiplex [9], threshold decomposition [6] and bit-wise elimination [1,8,11]. All can be used with the currently available, powerful FPGA devices. Still, the bit-wise elimination methods seem most appropriate for the cell array organisation.

Some specific positional filters have commercial VLSI implementations. There is no device that can be configured to realise any position filtering, though. Even if only median, min or max filtering is required it may be advantageous to use FPGA devices, as they offer greater versatility and ease of reengineering. Of course, FPGA implementations are particularly well suited for application in experimental image processing systems.

First attempts to use FPGAs as reconfigurable image filters were reported almost as soon as the devices became available [2,3,5]. The devices proved to be too inefficient for full-fledged use forcing the designers to limit the window size, pixel rates or the width of their bit

representations. This is no longer the case since the XC4000E family of devices became available [4].

Filter reconfiguration can be fully utilised only if there is an easy route to obtain new configuration variants. In case of FPGA implementation this is offered by auto-synthesis: new algorithms are described in terms of a hardware description language and the rest is done by the design tools with no human interaction. The results in the paper were obtained using the Xilinx Foundation tools with FPGA Express.

2. BITWISE ELIMINATION METHOD

Positional filtering is based on reordering of the pixel values according to their magnitude. Let's denote the *k*-th value in the reordered sequence by P_n^k (where *n* is the length of the sequence). After reordering, only a single value at the specific position is of interest. In bit wise elimination, values that are certain not to be at this position are removed from the sequence.

Values are compared bit-wise starting from the highest order bits. Lets assume that the *r*-1 highest order bits have already been analysed by this method. Then, all the values that are left for consideration must have the high order *r*-1 bits equal to each other (and to the result under evaluation). Values that had these bits different were eliminated leaving only *n'* values in the sequence. The position also has to be adjusted from initial *k* to *k'* after eliminating values that were greater. The *r*-th bit of result is determined as $P_{n'}^{k'}(r)$ by ordering the corresponding bits of the reduced sequence and considering *k'*-position. All the values that differ on the r-th bit from $P_{n'}^{k'}(r)$ are eliminated and *n'* is modified accordingly. If the eliminated values are greater than the quantile bit, then *k'* is also modified.

The algorithm ends when there is only one value left (or all the values left are equal to each other).

The approach, with changing *k* and *n* is not well suited for circuit implementations. Instead of eliminating the values, it is more convenient to modify them in a way that is guaranteed not to change the result [2,8,11]. If one knows that a value is larger than the *k*-quantile, all its lower bits are set to 1. If one knows that it is smaller, the lower bits are set to 0. Thus, single bit voting may still be used and the values of *k* and *n* are fixed. This is the method used for the presented FPGA implementations. It can be formally described by the following iterative equations, where iterations start with the highest order bits (*r*=*m*-1) and end with the lowest (*r*=0):

$$
\begin{aligned}
&M_i(r-1) = M_i(r) \vee \left(P_n^k(r) \oplus x_i(r)\right), \\
&S_i(r-1) = \left(M_i(r) \wedge S_i(r)\right) \vee \left(!M_i(r) \wedge x_i(r)\right), \\
&M_i(n) = S_i(n) = 0, i = 0..n-1, \\
&P_n^k(r) = \sum_{i=0..n-1} \left\{\left(!M_i(r) \wedge x_i(r)\right) \vee \left(M_i(r) \wedge S_i(r)\right)\right\} > k
\end{aligned}
\qquad (1)
$$

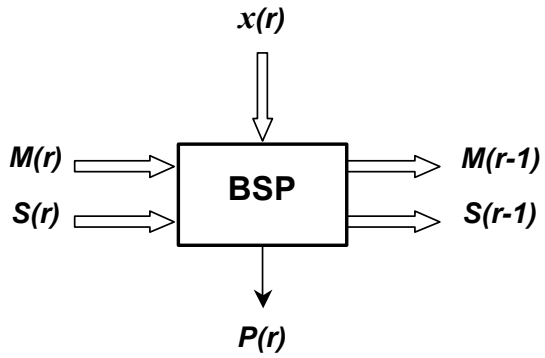

Fig. 1. Bit-slice processor

where $x_i(r)$ is the r-th bit of i-th pixel in the filtering window,
$P_n^k(r)$ is the r-th bit of the value at k-th position (k-quantile),
$M_i(r)$ and $S_i(r)$ are the modifying functions.

Using the presented equations (1) a bit-slice processor may be implemented (Fig. 1). This is a combinatorial circuit that processes the single bits of the input pixels to produce a single bit of the result. The most important part of this bit-slice is the thresholding function corresponding to the last of equations (1).

3. PIPELINED FILTER IMPLEMENTATIONS

The simplest hardware implementation of the filter can be obtained by using m bit-slice processors with connected modifying function inputs and outputs. This would be a fully combinatorial implementation with very long delays, as the modifying functions have to propagate from the highest to the lowest order bits.

Inserting pipelining registers between the bit-slice processors shortens the propagation paths [4]. The registers may be inserted either between every processor, as shown in Fig. 2, or only between some of them. Since this introduces latency between the bit evaluations, additional shift registers are needed on the inputs and outputs to ensure in-phase results.

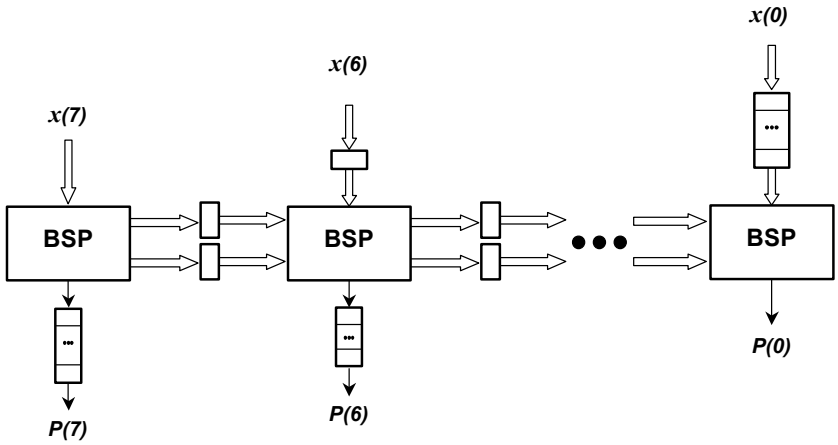

Fig. 2. Pipeline filter architecture

This architecture has very short propagation paths between registers and so it ensures highest pixel processing rates. There is latency between the input signals and the output equal to the number of bits in the pixel representations. Normally, in image processing applications this is not a problem. Just the image synchronisation signals need to be shifted correspondingly. It may be unacceptable, though, if image filtering is just a stage in a real-time control application.

4. FPGA IMPLEMENTATION RESULTS

The pipelined filter architecture was implemented for a filtering window of 3×3 pixels. The inputs of the filter were 3 pixel streams: one obtained by scanning the image and two delayed (by one and two horizontal scan periods). The consecutive horizontal window values were obtained by registering the input streams within the filter (to reduce the demand on input/output pads).

All the presented results were obtained by implementing the filter that computed the median. This has no significant effect on the device performance or complexity, except that the min and max filters have much simpler thresholding functions.

The filters were implemented using different size of pixel value representations (binary values of 4, 8 12 and 16 bits). In each case the smallest and fastest device that could contain the circuit was chosen for implementation. Table 1 contains the results of filter implementations using XC4000E family of devices, whereas Table 2 presents those for the Virtex-2 packages.

Table 1. Filter implementations using XC4000E devices.

Pixel representation	*Device*	*Used CLB's*	*Pixel rate*
4 bits	**4003EPC84-1**	**94**	**55.9 MHz**
8 bits	**4008EPC84-1**	**288**	**50.1 MHz**
12 bits	**4020EHQ208-1**	**645**	**50.6 MHz**
16 bits	**4025EPG223-2**	**993**	**37.5 MHz**

Table 2. Filter implementations using Virtex-2 devices.

Pixel representation	*Device*	*Used slices*	*Pixel rate*
4 bits	**2V40FG256-4**	**99**	**88.8 MHz**
8 bits	**2V40FG256-4**	**209**	**91.9 MHz**
12 bits	**2V80FG256-4**	**345**	**88.7 MHz**
16 bits	**2V80FG256-4**	**453**	**83.7 MHz**

The circuit complexity, expressed in terms of the number of cells used (CLB's or Virtex slices), results from the number and complexity of bit-slice processors (complexity of the combinatorial logic) and from the number of registers used in pipelining. The first increases linearly with the size of pixel representation. On the other hand the number of registers used in pipelining increases with the square of this representation. In case of the XC4000 architecture, the pixel representation of 8 bits is the limit, above which the complexity of circuit is determined solely by the pipelining registers (all the combinatorial logic fits in the lookup tables of cells used for pipelining).

The synthesis tools had problems in attaining optimal solutions for the synthesis of thresholding functions in case of the cells implemented in XC4000 devices (this was not an

issue in case of min and max positional filters). Most noticeably, the design obtained when the threshold function was described as a set of minterms required 314 CLB's in case of 8-bit pixel representation. By using a VHDL description that defined the function as a network of interconnected 4-input blocks, the circuit complexity was reduced to the reported 288 cells. The reengineered threshold function had slight effect on the complexity of 12-bit filter and none on the 16-bit one.

The most noticeable improvement in using the Virtex-2 devices for positional filter implementations was in the operation speed: approximately 50 MHz in case of the XC4000E devices and 80-90 MHz in case of Virtex-2. Some other architectural improvements are apparent, too. The increased functionality of Virtex slices led to much more effective implementations of pipelining registers: the FPGA Express synthesiser implemented them as shift registers instead of unbundled flip-flops, significantly reducing the slice usage. Improved lookup table functionality eliminated the problem of efficient decomposition of threshold function, too (at least in case of the 3×3 filtering window).

5. CONCLUSIONS

The presented implementation results show that FPGA devices have attained the speed grades that are more than adequate for implementing positional image filters of very high resolution. Furthermore, it is no longer necessary to interconnect multiple FPGA devices or limit the circuit complexity by reducing the pixel representations. In fact, the capabilities of Virtex-2 devices exceed these requirements both in terms of performance and cell count.

The proposed bit-wise elimination algorithm with pipelining is appropriate for the cell architecture of FPGA devices. The only problem is the latency, which may be too high in case of long pixel representations. By limiting the pipelining to groups of 2, 3 or more bit-slice processors it is possible to trade off latency against performance.

Positional filtering is just a stage in complex image processing. The analysed filter implementations leave a lot of device resources unused. This is so, even though the cell utilisation for representations of 8 bits or more is between 60 and 97%. The cells are mostly used for registering and the lookup tables are free. These may well be used to implement further stages of image processing.

It is very important that the considered implementations were directly obtained by synthesis from functional descriptions, expressed in VHDL language. This makes feasible the concept of reconfigurable filters, where the user describes the required filtering algorithms in a high-level language and these are programmed into the filter. Still, the design tools have not reached the desirable degree of sophistication and reliability. This is especially true of the obscure template matching rules, peculiar to specific synthesis tools. Also, the correctness by design paradigm is not always met – some errors of improperly matched templates were detected only by testing the synthesised device.

REFERENCES

[1] M.O.Ahmad, D.Sundararajan, "A Fast Algorithm for Two-Dimensional Median Filtering", *IEEE Trans. Circuits and Systems, 34(11)*, pp.1364-1374, 1987

[2] D.Caban & J.Jarnicki, “A Reconfigurable Filter for Digital Images Processing” (in Polish), *Informatyka, 6*, pp.15-19, 1992

[3] D.Caban, “Hardware implementations of a real time positional filter”, *Proc. 5th Microcomputer School Computer Vision and Graphics*, Zakopane, Poland, pp.195-200, 1994

[4] D.Caban & W.Zamojski, “Median filter implementations”, *Machine Graphics & Vision, 9(3)*, pp.719-728, 2000

[5] S.C.Chan, H.O.Ngai & K.L.Ho, “A programmable image processing system using FPGAs”, *Int. J. Electronics, 75(4)*, pp.725-730, 1993

[6] J.P.Fitch, E.J.Coyle & N.C.Gallagher, “Median Filtering by Threshold Decomposition”, *IEEE Trans. Acoustics, Speech and Signal Processing, 32(6)*, pp.553-559, 1984

[7] M.Juhola, J.Katajainen & T.Raita, “Comparison of Algorithms for Standard Median Filtering”, *IEEE Trans. Signal Processing, 39(1)*, pp.204-208, 1991

[8] C.L.Lee & C.W.Jen, “Binary Partition Algorithms and VLSI Architecture for Median and Rank Order Filtering”, *IEEE Trans. Signal Processing, 41(9)*, pp.2937-2942, 1993

[9] S.Ranka & S.Sahni, “Efficient Serial and Parallel Algorithms for Median Filtering”, *IEEE Trans. Signal Processing, 39(6)*, pp.1462-1466, 1991

[10] D.S.Richards, “VLSI Median Filters”, *IEEE Trans. Acoustics, Speech and Signal Processing, 38(1)*, pp.145-153, 1990

[11] C.-W. Wu: Bit-Level Pipelined 2-D Digital Filters for Real-Time Image Processing. *IEEE Trans. Circuits and Systems for Video Techn., 1(1)*, pp.22-34, 1991