HAL
open science

# Stream implementation of serial morphological filters with approximated polygons

Jan Bartovsky, Eva Dokladalova, Petr Dokládal, Vjaceslav Georgiev

## To cite this version:

Jan Bartovsky, Eva Dokladalova, Petr Dokládal, Vjaceslav Georgiev. Stream implementation of serial morphological filters with approximated polygons. 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Dec 2010, Athènes, Greece. pp.706-709, 10.1109/ICECS.2010.5724610 . hal-00622491

**HAL Id: hal-00622491**

**https://hal.science/hal-00622491**

Submitted on 17 Apr 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Stream Implementation of Morphological Filters with Approximated Polygons

Jan Bartovsky, Eva Dokladalova
UMR 8049 LIGM
A3SI, ESIEE Paris
University Paris-Est, France
{*bartovsj,dokladae*}*@esiee.fr*

Petr Dokladal
Dept. Mathematics
and Systems, CMM
Mines PARISTECH, France
*petr.dokladal@mines-paristech.fr*

Vjaceslav Georgiev
Electrical engineering,
Univ. of West Bohemia,
Czech Republic
*georg@kae.zcu.cz*

## Abstract

*This paper describes fast, stream implementation of serially composed morphological filters using approximated flat polygons.*

*The underlying 1-D dilation algorithm has interesting properties such as strictly sequential access to the input and output data and minimal memory and latency. These properties also apply to 2-D dilations and erosions by polygons, and consequently to serially composed advanced filters such as ASF or granulometries.*

*Implemented on a dedicated FPGA, this algorithm allows to obtain previously unequaled performances, opening the access to arbitrary size ASF or granulometries in time-critical, high-end (and also embedded) applications.*

## 1 Introduction

In the mathematical morphology, many modern practical applications rely on computational intensive morphological operators like compound filters (i.e. Alternate Sequential Filters), multi-scale analysis or object classification where multiple descriptors have to be evaluated. Such operators uses long concatenations of fundamental morphological operations of dilations and erosions with progressively changing structuring element (SE). The latency and memory requirements increase proportionally with the number of used dilations/erosions. Knowing that the latency and memory are functions depending on the SE and/or the image size, they can become a crucial problem for time critical applications [3].

Obviously, the above mentioned operators require to be able to compute efficiently different size of structuring elements. Nonetheless, not all of them can be easily implemented in real time with reasonable memory requirements. They remain out of reach for embedded systems with constraints on memory usage, computing latency and for the high definition video stream processing. Another problem is that modern applications require to evaluate not only a large scale of SE size but also a variety different SE shape. If we can illustrate numerous fast implementations, they remain dedicated to a given SE shape. On the other hand, if the algorithms allows to exploit various shapes, it does not necessarily respect the requirements on deep pipeline of compound operators.

The most efficient approaches are based on the structuring element decomposition. The acceleration of the algorithm is done within this reduced SE scope. The special attention is paid to the 1-D algorithms allowing to obtain a significant gain in overall performance. The most popular 1-D algorithm is the van Herk [7]. Although the computation complexity is independent of SE (O(1)), it requires two passes of input data: direct and reverse. This algorithm has later been improved Gil and Kimmel [4]. Their algorithm allows to approach 1, 5 comparison per pixel. However, the computing latency depends on the SE size W and the memory requirements increase. Recently, Clienti [2] has proposed an interesting modification of the van Herk algorithm reducing the memory, implemented on a streaming HW architecture on FPGA.

The efficient method how to compute arbitrary shaped SE is shown in [6] Van Droogenbroeck and Talbot. They use a histogram to store the values within the span of the SE at some position in the image. During the translation of the SE over later image positions the histogram is updated by inclusion/deletion of the values of the entering/leaving points. The family of available shapes for the SE is arbitrary. On the other hand, using histogram makes that the input data have to be integers. The computing latency is dependent on the SE shape and the histogram update complexity. Another efficient algorithm has been recently proposed by Urbach and Wilkinson [5]. It computes the erosion/dilation by a flat, arbitrary-shape SE by using a LUT. The algorithm is fast but it requires a considerable memory for storing the LUT . The computing latency and stream processing depend on which method is applied to the 1-D statistics measurements.

Equally, despite the numerous existing hardware implementations, the problem of optimization of deep morphological operators pipe is not really studied in the context of optimal execution time, reduced working memory and minimal latency and this for large structuring elements with different shape. We can cite, Velten and Kummert [8] proposing a delay-line based architecture. If It supports arbitrarily shaped SEs but with quadratic complexity (O(W2)) and high memory requirements due to need to store all pixels which are to be reused in calculation latter. Clienti [2] can generate hexagonal SE but its HW implementation relies on neighborhood processor; hence the large SE are obtained by homothecy and the latency and memory requirements increase.

*Principal contribution*

This paper describes fast, stream implementation of serially composed morphological filters using approximation of flat polygons. It is based on optimal memory, zero latency algorithm published in [1]. The proposed approach is based on strictly sequential access to the input and output data and it computes the results in only one scan of image. We show, how to apply this algorithm to generate 2-D dilations and erosions by polygons, and consequently to serially composed advanced filters such as Alternate Sequential Filters or granulometries. Implemented on a dedicated FPGA, this algorithm allows to obtain previously unequaled performances, opening the access to arbitrary size ASF or granulometries in time-critical, high-end (and also embedded) applications. The properties of SE: size and center can be modified in the run time in order to allow easy re-use of the proposed hardware block inside of one application.

*Paper organization*

The remainder of the paper is organized as follows. The Section 2 introduces mathematical background. The Section 3 outlines the structuring element decomposition mechanism and presents the algorithm principle. The proposed hardware implementation is discussed in Section 4. Finally the experimental results containing obtained performances are demonstrated in Section 5.

## 2 Basic notions

Let $\delta_B$, $\varepsilon_B$: $\mathbb{Z}^2 \rightarrow \mathbb{R}$ be a dilation and an erosion on grey-scale images, parameterized by a structuring element $B$, assumed rectangular, flat (i.e. $B \subset \mathbb{Z}^2$) and translation-invariant, defined as

$$\delta_B(f) = \bigvee_{b \in B} f_b \quad (1)$$

$$\varepsilon_B(f) = \bigwedge_{b \in \widehat{B}} f_b \quad (2)$$

The hat $\widehat{}$ denotes the transposition of the structuring element, equal to the set reflection $\widehat{B} = \{x \mid -x \in B\}$, and $f_b$ denotes the translation of the function $f$ by some scalar $b$. The SE $B$ is equipped with an origin $x \in D$.

## 3 Algorithm description

The algorithm principle uses the fact that Morphological dilation/erosion is separable into lower dimensions. For example, a rectangular SE can be separated into two parts: horizontal and vertical dilation by a segment. The 2-D result is obtained by concatenation of the two parts.

### 3.1 1-D Dilation algorithm

The implementation of Eqs. 1 and 2 consists of searching the extremum of $f$ within the scope of $B$

$$[\delta_B(f)](x) = \max_{b \in B} [f(x-b)] \quad (3)$$

and

$$[\varepsilon_B(f)](x) = \min_{b \in B} [f(x+b)] \quad (4)$$

The algorithm used below is based on the property [3] that for some $B(x)$ (which contains its origin) computing the dilation $\delta_B f(x)$ needs only those values of $f(x_i)$ that can "be seen" from $x$ when looking over the topographic profile of $f$, see Fig. 1. The values shadowed by the mountains can not be maxima, are exluded from the computation, and do not even be stored.

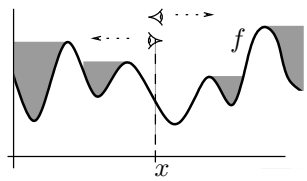


**Figure 1. Computing the dilation $\delta_B f(x)$: Values in valleys shadowed by mountains when looking from $x$ over the topographic relief of $f$ are useless. Notice that the masked values only depend on $f$, and not on $B$.**

From the implementation point of view, assuming a sequential access to the input data $f$, the dilation $\delta_B f(x)$ depends of points read after $x$. We say that $B$ is non causal. One can transform a non causal SE to a causal SE by utilizing the property that dilation commutes with translation.

$$\delta_{B+t} f(x) = \delta_B f(x-t) \quad (5)$$

where $t \in D$.

Note that the translation basically consists of writing the result to the correct place in the output stream.

These two principles are used by the Algorithm 1, see Appendix. For some input signal $F$, the algorithm computes the value $\delta_B F(wp) = F(rp)$. The coordinates $wp$ and $rp$ stand for the current *writing* and *reading* positions.

The Algorithm 1 is to be called from an outer loop iterating over the writing position in $\delta_B f$, such as while wp<N. The writing position wp is to be incremented whenever the Algorithm 1 outputs a valid value.

The input signal $f$ is sequentially read. A newly read value $F = f(rp)$ is inserted in the queue. The queue is a FIFO-like structure storing pairs of values $\{F, rp\}$, the sample F and reading position rp (see code line 5).

The useless values (that appear to not be maxima) are dequeued (code lines 1-2). Consequently, the values stored in the queue are always ordered in a decreasing order.

The values older than the current writing position $wp$ minus the length of $B$ are retrieved from the queue (code lines 3-4). The result of the dilation $\delta_B(x)$ is read at the front of the queue (code line 7). The result becomes available as soon as enough input data have been read, otherwise the output is empty (code line 9).

The next section explains the implementation of the algorithm from the following aspects:

### 3.2 2-D Dilation decomposition

**TO DO: Petr**

The separability of n-D morphological dilation into lower dimensions is a well known property. For example, a dilation by a rectangle can be obtained by dilating by a horizontal and then by a vertical segment. Preliminary results of fast, stream implementation of morphological filters (with rectangular structuring elements) have been published in [1].

To improve the angular isotropy one often prefers using circles approximated by polygons instead or rectangles. Regarding the implementation aspects, polygons are easily decomposable into obliquos segments.

$$\delta^{\text{2-D}} = \delta^{\text{1D}} \circ \ldots \circ \delta^{\text{1D}}$$

Previous implementations, see Soille [], accelerate the 1-D dilation algorithm. Computing the complete dilation by a polygon requires several iterations over the image. For serially composed filters such as ASF, granulometries etc. this requires a considerable number of iterations, which results in increasing time and latency. This rapidly becomes penalizing (or even an excluding) factor for using these operators in time-critical applications.

The Alg. 1 can be used for decomposition of polygones with interesting properties.

Tady chyby asi milion textu.

Priklad diskretizovanych SE a jejich popis, ktery se pouziva dal. Kazdy SE muze byt jinak velkej.

Problem SE pod 60ti stupni a jejich variance.

Aby SE vychazel u kraju tak, jak ma vypadat, musi se pridat border.

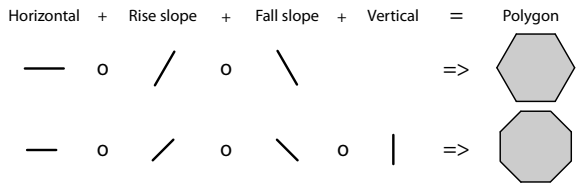


**Figure 2. Polygon SE decomposition.**

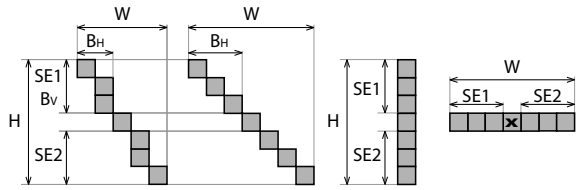


**Figure 3. Discrete SE description.**

## 4 Hardware Implementation

### 4.1 1-D Algorithm Implementation

The aforementioned Algorithm 1, along with the border and shifted addressing issues of outter controlling loop, is seen as a simple Mealy Finite State Machine (FSM). This FSM controls all the algorithm operations over the queue, position and value conditions etc.. It consists especially 2 main states $\{S1, S2\}$ and one auxiliary state $EOL$. The basic behavior of the direct algorithm implementation can be found at []. As a short reminder, the $S1$ state manages the *Dequeue useless values* and *Enqueue current sample* of the algorithm, and the $S2$ state handles the rest.

The auxiliary state $EOL$ is employed only when the end of image line is encountered. Its main purpose is to update the Offset value which determines a shift of the Queue numbering at the each image line. By its appropriate handling, i.e. incrementig, decrementing or holding still, it ensures an option to calculate dilation in inclined and programmable directions.

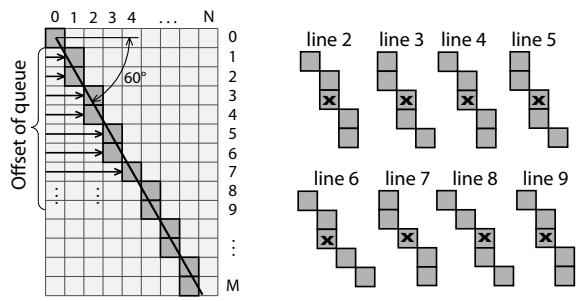


**Figure 4. Discrete slope SE.**

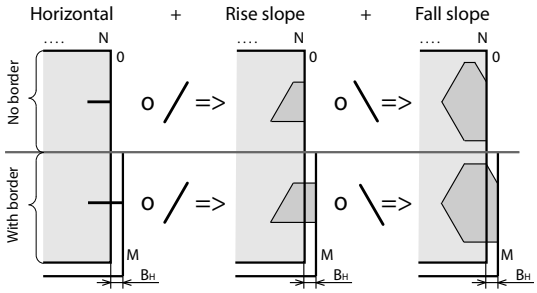


**Figure 5. Polygon SE composition with and without border.**

## 4.2 1-D Block Architecture

The Fig. 6 represents the HW architecture of the 1-D algorithm capable to operate in any of four potential directions: horizontal, vertical, rise or fall slope.

It can be separated into 2 areas, the FSM part and the memory part. The FSM manages the entire computing procedure and temporarily stores values in the memory. The memory instantiates one FIFO-like queue in the case of horizontal direction (horizontal scan), N queues in the vertical case (N is the image width), or N+W queues in the slope case (W is the width of slope SE) The queues are addressed by a modulo N, or modulo N+W respectively, Page counter (inactive in the case of horizontal direction).

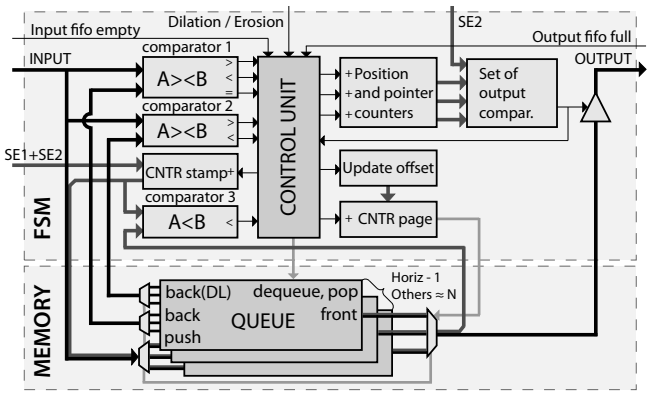


**Figure 6. Overview of implemented 1-D architecture. The FSM part manages computation, memory part contains the data storage–queues**

The Control unit is a sequential circuit that manages the state transitions. It increments the rp, wp, Page and position Stamp counter (modulo $SE1 + SE2$) appropriately. The Control unit also operates the queue memory operations and the back-pressure full flag used for the data-flow control.

The processing of one pixel proceeds as follows: In the beginning of $S1$, the last queued pixel is invoked by Back() operation from queue and fetched to the Comparator 1 where it is compared with the current sample. The Control unit decides on the basis of comparison results and selected morphological function whether the enqueued pixel is to be dequeued (lines 1-2). Otherwise, the current pixel is extended with position stamp and enqueued (line 3).

The $S2$ invokes the oldest queued pair {pixel, stamp} by the Front() operation. This read pixel is outputed as a correct result if the set of output conditions (code line 6 for horizontal and vertical unit, and plus border conditions for both slope units) is fulfilled. The deleting of too old values is performed by comparing the reading-position stamp with the actual one. Notice that eventual deletion has no impact on output dilation value because the Pop() operation (lines 4 and 5) issued by the Control unit goes into effect with a next clock edge.

According to the need of shifted queue addressing in the case of rise or fall slope unit, a several modification have been done on the preceding version [].

Since the queues are in the vertical-like direction, they can be addressed by the Page counter which is incremented with every received pixel of horizontal scan. If the initial value of Page counter remains still in every line, its value will be same in any column over all lines leading into purely vertical dilation. But if the initial value of Page counter at the begining of image line varies according to intended angle of inclination, all queues will follow the same direction. The initial value of Page counter is called Offset.

The Offset is updated at the end of every image line on basis of one iteration of Bresenham's line algorithm that is implemented as shown at Fig. 7. Note that a change of either modulo or step constants enable us to adjust the inclination angle of SE. This unit can handle any angle the tangent of which can be expressed as rational number within given precision. For instance we use modulo 26 and step 15 for hexagon SE because the angle is equal to $\arctan(15/26) = 29.98°$. The sense of slope is defined by the Offset counter direction; up-counting for a fall slope and down-counting for a rise slope.

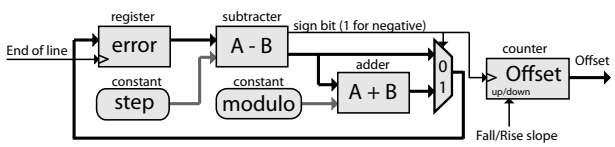


**Figure 7. Update offset block inner scheme.**

The entire set of parameters, i.e. image width and height, SE dimensions and morphological function select, is runtime programmable at the beginning of the frame.

### 4.3 Stream Processing

It is well known that dilations are separable into lower dimensions. Hence, dilation with polygons can be obtained by cascading several 1-D dilations. This architecture allows to achieve this separation without violating the sequential access to data and without intermediate data storage.

The single dilation/erosion block for hexagonal or octagonal SE is shown at Fig. 8. It can be divided into three different purpose parts: controller, computation and border handling part.

The controller is in charge of whole system behavior. It recieves the SE diameter and shape select signal, then it deduces particular SE sizes for every computing unit and borders from them, and initiates the computation. When the frame has been completely processed, the superior system is acknowledged. The controller held all inner blocks in the idle state as long as they are not utilized.

The border issue of polygonal SE is managed by a pair of dual blocks. At the front end, the border of computed size containing only the recessive pixel value (considering 8-bit grayscale 0 for dilation, 255 for erosion) is added at the Add border while the border is deleted before the output port at the Remove border block. The necessity along with depths of input and output FIFOs is to be figured out eventually considering the input/output stream frame timings. In the case the video blank areas (synchronization, blak porches etc.) is greater then added border, the input FIFO can be omitted.

The computation part mainly consists four previously described 1-D computation units with distinct preconfiguration. Although this 1-D unit (as represented by Fig. 6) is capable to work in any of four intended directions, its instances are tailored to a single direction during synthesis for proper resources consumption. Taking place in a simple pipeline, the output of each unit is read by successive unit immediately at the next clock cycle.

Note that the output of every computation unit is a partially processed scan-ordered data flow which can be used for multi-scale analysis. Then the dilation with linear, rectangular and octagonal SEs (all centered) can be obtained during a single image scan (considering units re-ordering).

For reading and writing in a stream, one needs to handle different clock rates. Although two equidistant input and output streams have a constant rate - 3 clk/px or more, the algorithm latency to compute the dilation for one pixel varies. To handle the latency difference, one needs to use balancing FIFOs. The depths of these FIFOs directly define the maximal inserted delay by our block that has to be accepted as a permanent delay between the input and output streams. Obviously, the FIFOs should be as small as possible.

A balancing FIFO is filling when a precedent unit outputs data faster than a successive unit can read. The dequeuing worst-case defines the necessary depth as follows

$$F_{hor} = \frac{Steps_{hor} + 2}{StreamRate} - 1 \tag{6}$$

$$F_{ver} = N\left(\frac{Steps_{ver} + 2}{StreamRate} - 1\right) \tag{7}$$

$$F_{slope} = (N + W)\left(\frac{Steps_{ver} + 2}{StreamRate} - 1\right) \tag{8}$$

where $Steps_x$ is the number of the dequeue steps.

The purpose of output FIFO is to ensure the permanent stream delay in all circumstances, if necessary. Its depth is a maximal depth of balancing FIFOs. The instantaneous filling of the output FIFO is complementary to filling of all balancing FIFOs combined, and thus the overall delay of entire chain is not augmented. Furthermore, if more polygonal blocks are pipelined to form compound operators (e.g. opening, closing, ASF), only one output FIFO at the end is necessary.

### 4.4 Clock rate

The overall average clock rate stays in the interval from 2 clock cycles per pixel in the best case when the image is constant, up to 3 clock cycles per pixel in the worst case. The real rate between 2 and 3 clock cycles per pixel depends on the picture contents.

Note that a locally worst case is encountered when a monotonously decresing signal is followed by a high upstep. Here, the latency locally increases due to that a number of samples need to be dequeued at a time (the S1 state of the FSM). However, this local latency increase is globally compensated by the fact that during the entire monotononous portion of the signal no values have been dequeued. Hence, the average per pixel clock cycles remains unchanged.

### 4.5 Memory Requirements

The memory requirements of proposed architecture consist of the computation-involved block of queues and the balancing FIFO (defined at (6), (7) and (8)) for every computation unit, and input/output buffers if necessary.

Despite the algorithm description considers separated queues as a storage platform, they are gathered into a single dual-port memory within every unit from three reasons: 1) only one queue is accessed at a time while the others remain idle, 2) instantiating N separated memories would be resource inefficient because the FPGA RAM blocks could not be exploited, 3) single memory block allows to use an off-chip memory. Hence all the queues are mapped side by side into a single linear memory space, see Fig. 9. Every queue

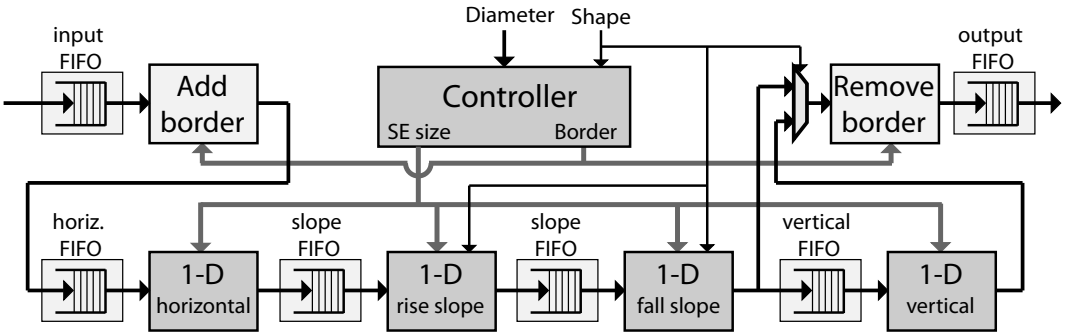


**Figure 8. Overall architecture of 2-D**

has a related pair of front and back pointers which must be retained throughout entire computation process. The appropriate pair is always read before the particular queue is used as well as the modified pointers are stored back after the computation left the queue. These pointers are stored in a separated pointer memory. For a price of small memory increment, the queues can be usefully packed into RAM blocks.

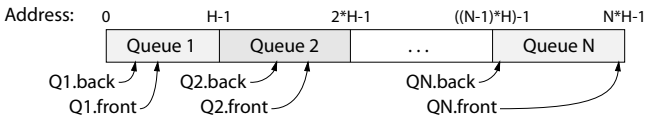


**Figure 9. Vertical Queues are mapped into linear memory space side by side.**

The respective unit memory contributions are (note that W, H are width and height of SE, bpp stands for bits per pixel):

$$M_{hor} = W(bpp + \lceil \log_2(W-1) \rceil) \quad \text{[bits]} \tag{9}$$

$$M_{ver} = N(H(bpp + \lceil \log_2(H-1) \rceil) + 2\lceil \log_2(H-1) \rceil) \quad \text{[bits]} \tag{10}$$

$$M_{slope} = (N+W)(H(bpp + \lceil \log_2(H-1) \rceil) + 2\lceil \log_2(H-1) \rceil) \quad \text{[bits]} \tag{11}$$

For instance, the SVGA image and octagonal SE 71x71 px has the minimal required memory $M_{hor} + M_{ver} + 2 \cdot M_{slope} \approx 795$ kbits (for stream plus balancing FIFOs of $F_{hor} + F_{ver} + 2 \cdot F_{slope} \approx 56$ kbits). For SE 41x41 px it is only $\approx 456$ kbits, and $\approx 33$ kbits of FIFOs respectively, which is far below the size of input image $800 \cdot 600 \cdot 8 = 3.84$ Mbits.

## 5 Experimental results

The proposed polygonal stream processing architectures has been implemented in the VHDL using the XST synthesis tool, and targeted to the Xilinx FPGA Virtex-5 (XC5VLX50T-1FF1136). The resulting clock frequency is 100 MHz. The algorithms queues are gathered in a block RAM memory, and thus its access time augment the critical path delay.

The experimental results in terms of latency, number of frames per second and average processing rate are outlined at Table 1 for various image resolutions. The SEs are either hexagon or octagon with diameter of 51 px. One can see that latency is proportinal with regard to the SE size $\times$ image width $\times$ pixel rate whereas the average pixel rate remains constant. The small variation of pixel rate, as well as the latency augmention, is caused by added border the size of which depends on the size of SE.

**Table 1. Timing, frame rate w.r.t. image size, diameter of SE = 51 px.**

| Size of Image | CIF | VGA | SVGA | XGA |
|---|---|---|---|---|
| Lat. Hex [clk] | 28082 | 49459 | 60799 | 78513 |
| Rate Hex [clk/px] | 2.448 | 2.412 | 2.397 | 2.410 |
| Exper. FPS Hex | 344 | 123 | 80 | 49 |
| Lat. Oct [clk] | 25976 | 46474 | 57357 | 74273 |
| Rate Oct [clk/px] | 2.517 | 2.452 | 2.431 | 2.436 |
| Exper. FPS Oct | 354 | 125 | 81 | 50 |
| Stream FPS | 282 | 99 | 64 | 39 |

The Table 2 presents an influence of varying SE size on the timing properties of proposed architecture. As mentioned before, the larger SE invokes the larger latency and border. As a consequence, both slightly degrade the computation throughput.

The average processing rate specifies a number of clock cycles needed for process one pixel of image including bor-

**Table 2. Timing, frame rate w.r.t. SE, SVGA image size.**

| SE diameter | 21 | 31 | 41 | 51 | 61 |
|---|---|---|---|---|---|
| Lat. Hex [clk] | 21822 | 32701 | 49420 | 60799 | 72459 |
| Rate Hex [clk/px] | 2.379 | 2.386 | 2.392 | 2.397 | 2.401 |
| Exper. FPS Hex | 85 | 83 | 82 | 80 | 79 |
| Lat. Oct [clk] | 23882 | 32968 | 47776 | 57357 | 66953 |
| Rate Oct [clk/px] | 2.398 | 2.408 | 2.421 | 2.431 | 2.442 |
| Exper. FPS Oct | 85 | 84 | 82 | 81 | 80 |

der. Its main intention is to enable a comparison of different image and SE results. This rate can be computed as (12), where $T_{proc}$ is overall time consumed by processing one frame and $B_H$, $B_V$ denotes border.

$$AR = \frac{T_{proc} - 2B_H M}{(N + 2B_H)(M + 2B_V)} \quad \text{[clk/px]} \tag{12}$$

Considering a frame rates, the experimental values are obtained directly from measured frame time $T_{proc}$ and clock frequency $f_{clk}$ (100 MHz). In addition, if the average processing rate is constrained (e.g. given by input stream rate), the resulting frame rate can be determined by (13).

$$FPS = \frac{f_{clk}}{AR(N + 2B_H)(M + 2B_V) + 2B_H M} \quad \text{[fr/s]} \tag{13}$$

The proposed stream architecture (of Fig. 8) has been synthetized for SE heights up to 61 px for hexagons or 91 px for octagons (virtually 75 px for regular octagons) operating with SVGA image. This configuration needs all computation units to provide SEs 31 px tall (wide for horizontal one). The FPGA area comprises of almost 4500 6-input LUTs, 1000 registers and 30 36-kbit blocks of on-chip SRAM memory.

Compared with our architecture, the recently proposed Clienti's SPoC (Several neighborhood Processors On Chip) yields a relative throughput of about 400 Mpx/s per single computation core with arbitrary SE 3x3 px. The larger SE are obtained by multiple consecutive passes regarding the SE composition technique. The performance is then decreased to e.g. 25 Mpx/s for SE 33x33 px which is worse than our 30 Mpx/s (experimentally up to 40 Mpx/s) with polygonal SE of arbitrary size (tested up to 91x91 px). In addition, thanks to used algorithm with constant complexity, our design exceeds Clienti's one on field of efficiency because he perform 8 comparisons per pixel for elementary 3x3 px SE.

# 6 Conclusions

This paper describes a fast, stream implementation of morphological dilations and erosions with polygons, improving thus angular isotropy with regard to previously used rectangles. The implementation conserves the same intersting properties, such as linear computational complexity w.r.t. image size and independent of the SE size.

The latency is strictly equal to the latency of the operator, i.e. it is inferred by the size of the used structuring element. The operator uses strictly sequential access to data.

The memory consumption is far below the size of the image which allows to embed on a single chip complex operators able to process large images useble for construction of more complex operators such as ASF, granulometries, etc. with the same properties and performance.

These performances allied to the possibility of SoC-embedding opens the usability of previously inaccessible advanced morphological operators such as ASF or granulometries in industrial systems running under severe time constraints.

# References


[1] J. Bartovsky, E. Dokladalova, P. Dokladal, and V. Georgiev. Pipeline architecture for compound morphological operators. In *ICIP*, September 2010.

[2] C. Clienti. *Data flow architectures dedicated to image processing using mathematical morphology*. PhD thesis, School of Mines Paris, Centre de Morphologie Mathmatique, Sept. 2009. ID:5758.

[3] P. Dokládal and E. Dokládalova. A zero-latency, optimal-memory algorithm for morphological operations. Technical Report N-01-2010, Center of Mathematical Morphology, Mines-PARISTECH, January 2010.

[4] J. Gil and R. Kimmel. Efficient dilation, erosion, opening, and closing algorithms. *IEEE Trans. PAMI*, 24(12):1606–1617, 2002.

[5] E. R. Urbach and M. H. F. Wilkinson. Efficient 2-d grayscale morphological transformations with arbitrary flat structuring elements. *IEEE Transactions on Image Processing*, 17(1):1–8, 2008.

[6] M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recogn. Lett.*, 17(14):1451–1460, 1996.

[7] M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recogn. Lett.*, 13(7):517–521, 1992.

[8] J. Velten and A. Kummert. Fpga-based implementation of variable sized structuring elements for 2d binary morphological operations. In *DELTA*, pages 309–312, 2002.

## Appendix: 1-D Dilation Algorithm Pseudocode

**Algorithm 1**: dF ← 1D_DILATION (rp, wp, F, SE1, SE2, N)

**Input**: rp, wp - reading/writing position; F - input signal value (read at rp); SE1, SE2 - SE size towards left and right; N - length of the signal; Q - a FIFO-like queue

**Result**: dF - output signal value (to be written at wp)

```
while Q.back()[1] ≤ F do
    Q.dequeue() ;                    // Dequeue useless valuess
Q.push({F, rp}) ;                    // Enqueue the current sample
if wp - SE1 > Q.front()[2] then
    Q.pop() ;                        // Delete too old value
if rp = min (N, wp + SE2) then
    return (Q.front()[1] ) ;         // Return valid value
else
    return ({}) ;                    // Return empty
```