Low-Complexity Binary Morphology Architectures with Flat Rectangular Structure Elements

Hedberg, Hugo; Kristensen, Fredrik; Öwall, Viktor

Published in:
IEEE Transactions on Circuits and Systems Part 1: Regular Papers

DOI:
10.1109/TCSI.2008.918140

2008

Low-Complexity Binary Morphology Architectures With Flat Rectangular Structuring Elements

Hugo Hedberg, *Student Member, IEEE*, Fredrik Kristensen, *Student Member, IEEE*, and Viktor Öwall, *Member, IEEE*

***Abstract*—This article describes and evaluates algorithms and their hardware architectures for binary morphological erosion and dilation. In particular, a fast stall-free low-complexity architecture is proposed that takes advantage of the morphological duality principle and structuring element (SE) decomposition. The design is intended to be used as a hardware accelerator in real-time embedded processing applications. Hence, the aim is to minimize the number of operations, memory requirement, and memory accesses per pixel. The main advantage of the proposed architecture is that for the common class of flat and rectangular SEs, complexity and number of memory accesses per pixel is low and independent of both image and SE size. The proposed design is compared to the more common delay-line architecture in terms of complexity, memory requirements and execution time, both for an actual implementation and as a function of image resolution and SE size. The architecture is implemented for the UMC 0.13-μm CMOS process using a resolution of 640 $\times$ 480. A maximum SE of 63 $\times$ 63 is supported at an estimated clock frequency of 333 MHz.**

***Index Terms*—Application-specific integrated circuit (ASIC), binary, dilation, erosion, field-programmable gate array (FPGA), hardware, image processing, morphology, surveillance system.**

I. Introduction

MATHEMATICAL MORPHOLOGY (MM) is a set of mathematical tools used to manipulate the shape or understand the structure of connected clusters of pixels [1]. MM are set-theory based methods of non-linear image analysis and plays an important role in many digital image processing applications, e.g., robot and computer vision, object recognition, and automated surveillance systems. The methods were originally developed for binary images, i.e., the 2-D integer space $\mathbb{Z}^2$, but were soon extended and now apply to several image representations, e.g., gray scale and various color spaces. However, since binary MM is used in many applications, it is of special interest to take advantage and exploiting the reduced complexity binary filtering offers, which requires effective algorithm and hardware architectures for this type of filtering.

Erosion (ε) and dilation (δ) are the two foundations in MM, since all morphologic operations can be broken down into these two basic operations [1]. For example, operations such as opening, closing, gradient, and skeletonization are performed with these two base functions. Thus, the need for low complexity architectures to perform ε and δ is evident.

Manuscript received May 9, 2007; revised November 2, 2007. First published February 8, 2008; current version published September 17, 2008. This work was supported by the Competence Center for Circuit Design at Lund University. This paper was recommended by Associate Editor A.-Y. Wu.

The authors are with the Department of Electrical and Information Technology at Lund University, Lind SE-221 00, Sweden (e-mail: hhg@eit.lth.se; fkn@eit.lth.se vikt@eit.lth.se; hugo.hedberg@eit.lth.se).

Digital Object Identifier 10.1109/TCSI.2008.918140

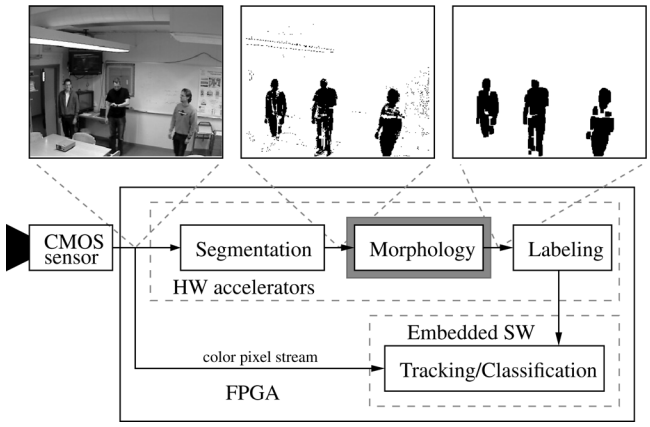

Fig. 1. Conceptual overview of an automated digital surveillance system of which the gray area is addressed in this paper. Input and output to the segmentation algorithm and the result after the morphological filter is also visualized.

There are numerous applications using different binary morphological operations reported in literature, e.g., noise filtering, boundary detection, and region filling [2]. The binary image representation can emerge not only due to the nature of the application, e.g., performing character recognition on a black and white document, but also as output from an image processing step, e.g., intensity thresholding, segmentation, or thresholded convolution [3], [4]. As a detailed example of such an application, a real-time automated surveillance system incorporated in a network camera is described by Kristensen *et al.* [5]. Since the system is intended for a self-contained camera, low complexity and low power consumption are main constraints due to limited hardware resources and to avoid heat problems. A conceptual overview of this system when implemented on a field-programmable gate array (FPGA) is shown in Fig. 1. The camera feeds the image processing system with a real-time image stream with 25 frames per second (fps). A segmentation algorithm [6] preprocesses the image stream and produces a binary motion mask in which zeros and ones corresponds to background and foreground, respectively. In theory, the moving parts of an image should be distinguished as independent objects in the binary mask. However, in reality the mask will be distorted with noise and single objects split into multiple clusters of connected foreground pixels. In order to remove noise and reconnect split objects, one or more opening (ε followed by δ) and closing (δ followed by ε) operations are performed on the mask. The result of an opening performed on the motion mask is shown in Fig. 1, where the noise has been reduced. The object classification part uses the mask to find the moving objects in the color image, extract features, and perform classification as well as tracking.

When designing a morphological hardware unit, there are many application specific questions and issues that have to be addressed, e.g., required class of supported structuring elements (SE) and issues related to the imposed bandwidth. Depending on the answers, certain trade-offs can be made in the architecture. However, there are some properties, apart from the obvious such as low complexity and fast execution time, that are advantageous and should be taken into consideration under any circumstance. First of all, most images are acquired and stored in raster scan order, i.e., starting in the upper left corner of the image and proceeding row by row down to the lower right corner. Therefore, to easily incorporate the units into this type of system environment and at the same time avoid intermediate data storage and random memory accesses during the pixel processing, the architecture should use the input and produce the output in raster scan order. This allows burst reads from memory and the possibility to place several units sequentially after each other (without intermediate storage). Secondly, all morphological operations are based on evaluating values contained in a local neighborhood of a pixel defined by the SE. Naturally, extracting and performing calculations on large neighborhoods can become particularly computationally intensive. Therefore, the main obstacle when designing a morphological hardware architecture is to extract this neighborhood and perform the calculation with minimal hardware resource utilization and still preserve raster scan order. Furthermore, to increase the flexibility and thereby overall system performance, it is desirable that the size of the SE can be changed during run-time. As an example, in the automated surveillance application [5], a flexible SE size can be utilized to compensate for different types of noise and to sort out certain types of objects in the mask, e.g., high and thin objects (standing humans) or wide and low objects (side view of cars).

A. Previous Work

Mathematical morphology is and has been a subject of extensive research resulting in numerous books and articles, covering both the theoretical and hardware aspects of this field. However, to put this work in perspective, only other important hardware architectures performing binary erosion and dilation are discussed here.

Fejes and Vajda [7] and Velten and Kummert [8] propose a delay-line architecture for 2-D binary erosion or dilation. This classical approach supports arbitrary shaped SEs, but the hardware complexity is proportional to N_{se}^2, where N_{se} is a side in the maximum supported square SE. The pixels that are to be reused are stored in delay-lines, resulting in a memory requirement proportional to $N_{\mathrm{se}}N$, where N is the image width. Therefore, this type of implementation becomes unsuitable for large SEs and high-resolution applications. In [9], an architecture using the same type of delay-lines is proposed, thus having the same memory requirement. However, based on the observation that many calculations between two adjacent pixels are redundant and partial results can be reused, the number of comparators per output value can be reduced to $2\lceil \log_2(N_{\mathrm{se}}) \rceil$ for certain SE shapes, e.g., rectangles.

Źarandy *et al.* propose a cellular neural network approach to perform binary erosion and dilation in [10]. It is shown that binary morphology apply to this type of structures but as for the delay-line architecture, the computational complexity is proportional to N_{se}^2. This is due that the SE is mapped onto an array of cells, in which each element requires a separate cell. In addition, since the pixels needs to be reused they need to be stored, preferable done in delay-lines, once more ending up with a memory requirement proportional to $N_{\mathrm{se}}N$. However, this approach opens up for other possibilities considering the learning feature of such networks to control the size and shape of the SE but this is not further addressed in this article.

Malamas *et al.* presents a fast systolic architecture performing a 1-D binary erosion or dilation (or a combination of these) in [11]. The architecture can be extended to 2-D by parallel processing of 1-D units. Processing each row in parallel makes it fast but the drawback of this architecture is that the complexity of each 1-D branch is proportional to the SE width making it unsuitable for applications requiring large SEs.

The implementation by Van Herk [12] support large 1-D (linear) SEs and performs each operation with only three comparisons per output pixel independent of the SE size. However, the implementation requires two scans to complete each 2-D operation, i.e., the 1-D filter first applied on rows then on columns, thus requiring intermediate storage to transpose the image. Although the implementation supports binary filtering, it is more applicable in gray scale applications.

In [13], a low-complexity and low memory requirement architecture performing a 2-D binary erosion or dilation that takes advantage of the morphological duality and SE decomposition is proposed. The main advantage of this architecture is that it has constant computational complexity, i.e., each output value is calculated with only 4 operations per pixel independent of the SE size, and a memory requirement proportional to $N \log_2(N_{\mathrm{se}})$. The class of supported SEs is limited to rectangles of arbitrary size. However, in this architecture, the input stream has to be stalled during padding which requires additional memory.

B. Main Contribution

The main contribution of this article is to present a stall-free hardware architecture for binary ε and δ, together with its evaluation and examples of its application context. Since the architecture is an extension of the one published in [13], they share many hardware properties, e.g., only supports rectangular SEs and requires the same number of operations per pixel. However, by parallel processing, the stall cycles during padding can be avoided, thus no additional memory is required. This modification results in an architecture that uses a single clock domain and is superior both in terms of computational complexity and memory requirement compared to the others discussed in Section I-A. For evaluation purposes, the delay-line architecture discussed above is presented and used as reference design. Hardware requirements, measured in complexity, memory, and execution time, for the different designs are compared both for an actual implementation and as a function of image resolution and SE size.

The remainder of this article is organized as follows: Section II gives an introduction to morphology and certain properties used in the proposed architectures. Section III discusses different hardware architectures for binary morphology

together with giving the context in which they can be used. Implementation results of these architectures are then evaluated in terms of computational complexity, execution time, and memory requirements in Section IV and Section V, respectively. The article is concluded in Section VI.

II. Morphology

All morphological operators are based on evaluating subsets, or neighborhoods, of connected pixels in the input frame I, determined by a kernel referred to as SE. The SE has two tasks: determine which pixels from I to include in the neighborhood and define the position of the output, O, in the resulting frame, which usually corresponds to the center position of the SE. Let the 2-D SE be flat, i.e., $\mathrm{SE}_{i,j} = 1 \; \forall i,j \in \mathrm{SE}$, and the input image binary, i.e., $I \in \mathbb{Z}^2$. The SE slides over I so that the superimposed O visits each coordinate in I once. This results in an output image with the same dimension as I and a content based on the evaluation of the subsets determined by the SE. In words, a binary ε produces a ONE at every position where the superimposed SE has the same geometrical shape as I, equivalent to a logical AND operation. A binary δ is the dual of an ε, i.e., a ONE is produced at all locations where the locally superimposed SE has at least one element equal to I, corresponding to a logical OR operation. Mathematically, the ε and δ of I by SE, denoted $I \ominus \mathrm{SE}$ and $I \oplus \mathrm{SE}$ respectively, are defined as

$$\varepsilon: \quad I \ominus \mathrm{SE} = \{z | (\mathrm{SE})_z \subseteq I\}$$
$$\delta: \quad I \oplus \mathrm{SE} = \left\{z | \left[(\widehat{\mathrm{SE}})_z \cap I\right] \neq \emptyset\right\}$$

and one is the dual of the other according to

$$I \oplus \mathrm{SE} = (I' \ominus \widehat{\mathrm{SE}})' \tag{1}$$
$$I \ominus \mathrm{SE} = (I' \oplus \widehat{\mathrm{SE}})' \tag{2}$$

where $'$ is bit inversion and $\widehat{\mathrm{SE}}$ is the reflection of SE [14].

Sliding window operations (including ε and δ) are prone to boundary problems. These occur when the SE reaches outside of the boundaries of I, as shown in Fig. 2(a). There are several methods to address this issue. Simply omitting these values from the calculation is the most straightforward. Another frequently used method is to insert extra pixels outside the image, which is called padding. This can be seen as temporarily increasing the resolution until a well defined output is obtained, as shown in Fig. 2(a). In case of ε and δ, the padding should not affect the output result and the inferred bits are therefore defined as ONES and ZEROS for ε and δ [14], respectively. With these definitions, information around the boundaries of I will not be corrupt since the output only depends on the image content.

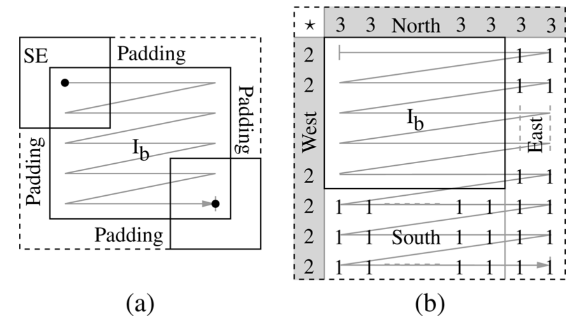

Fig. 2. (a) Iillustration of the boundary problem where the SE stretches outside of the image borders and where the padding is inserted. (b) Example of padding values using an $\mathrm{SE} = 7 \times 5$. $\star$ marks a *don't care* position. Padding to the east and south is inserted in the data stream which is shown by the gray arrow. The west and north padding are not part of the data stream and is only used as initial values for the memory.

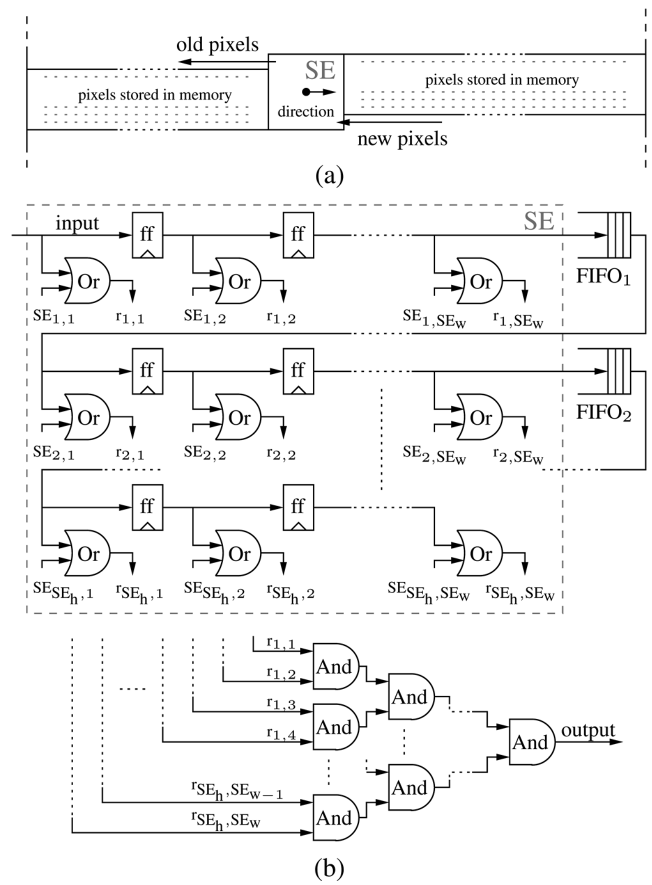

Fig. 3. (a) Illustration of the incoming pixel stream and pixels stored in memory in a direct mapped implementation. (b) Delay-line architecture of a morphological erosion block. The gray dashed line indicates the pixels covered by the SE. SE_h and SE_w is the height and width of the SE. Control logic is omitted for clarity.

III. Architecture

A. Delay-Line Architecture

The delay-line architecture is a direct mapping of the ε or δ operation [7], [15]. The main idea is to store pixels to the left and right of the SE as long as they are to be used in future output calculations, i.e., all consecutive pixels from the upper left corner to the lower right corner of the SE are stored in one long memory chain, as shown in Fig. 3(a). As an incoming pixel is shifted in, the oldest pixel currently in the architecture is shifted out. An example of such an architecture that implements this functionality for erosion is shown in Fig. 3(b), and is used as a reference throughout the paper. All pixels covered by the SE are stored in flip-flops and are thus individually accessible and all pixels in between two rows of the SE are stored in first in first out (FIFOs). This allows the SE to be moved to the next position by reading a new input and moving all other pixels one step in

Fig. 4. Decomposition of a flat SE of size 3×7 into $\mathrm{SE}_1 = 1 \times 7$ and $\mathrm{SE}_2 = 3 \times 1$.

the memory chain, achieved by either shifting data or changing memory pointers. The major benefit of this architecture is its ability to support streaming input data and arbitrary shaped SEs. Control logic to manage the enable signals for each element in the SE ($\mathrm{SE}_{x,x}$) and to change the morphological operation to dilation is omitted in Fig. 3(b).

In addition, to handle the boundary issue, the ability to control each position in the SE is used. The parts of SE that extend outside the image are forced to ONE in accordance with the definition, see Section II. Practically, the control signal $\mathrm{SE}_{x,x}$ in Fig. 3(b) is set to ONE for the parts of the SE that are outside of the image.

B. Low-Complexity Architecture

In order to improve the delay-line architecture in terms of operations per pixel and memory requirements, a low-complexity architecture based on decomposition and duality was initially proposed in [13].

The morphological operation δ is associative, which means that if the SE can be decomposed into smaller SEs according to

$$\mathrm{SE} = \mathrm{SE}_1 \oplus \mathrm{SE}_2 \tag{3}$$

and shown in Fig. 4, dilating I by SE gives the same result as when first dilating I with SE_1 and then dilating the result with SE_2 according to:

$$I \oplus \mathrm{SE} = I \oplus (\mathrm{SE}_1 \oplus \mathrm{SE}_2) = (I \oplus \mathrm{SE}_1) \oplus \mathrm{SE}_2. \tag{4}$$

With a decomposed SE, the number of comparisons per output pixel is decreased from the number of ones in the SE to the number of ones in SE_1 plus SE_2. As an example, using the flat SE in Fig. 4, the number of comparisons per output is decreased from 21 to 10.

If the SE is both reflection-invariant, i.e., $\mathrm{SE} = \widehat{\mathrm{SE}}$, and decomposable, the following two equations can be derived by combining (1) and (4)

$$\begin{aligned} I \ominus \mathrm{SE} &= I \ominus (\mathrm{SE}_1 \oplus \mathrm{SE}_2) = (I' \oplus (\mathrm{SE}_1 \oplus \mathrm{SE}_2))' \\ &= ((I' \oplus \mathrm{SE}_1) \oplus \mathrm{SE}_2)' = ((I'' \ominus \mathrm{SE}_1)' \oplus \mathrm{SE}_2)' \\ &= ((I \ominus \mathrm{SE}_1)'' \ominus \mathrm{SE}_2)'' = (I \ominus \mathrm{SE}_1) \ominus \mathrm{SE}_2 \end{aligned} \tag{5}$$

and

$$\begin{aligned} I \oplus \mathrm{SE} &= (I \oplus \mathrm{SE}_1) \oplus \mathrm{SE}_2 = (I' \ominus \mathrm{SE}_1)' \oplus \mathrm{SE}_2 \\ &= ((I' \ominus \mathrm{SE}_1)'' \ominus \mathrm{SE}_2)' = ((I' \ominus \mathrm{SE}_1) \ominus \mathrm{SE}_2)'. \end{aligned} \tag{6}$$

Comparing (5) and (6), it can be seen that both ε and δ can be expressed as an erosion (or as a dilation). This property is known as the duality principle. Finding decompositions to an arbitrary SE is a difficult problem and not always possible [16]–[18]. In addition, for an SE to be reflection-invariant, it has to be symmetric with respect to both x- and y-axes, e.g., an ellipse. However, a common class of SEs that is both decomposable and reflection invariant is flat rectangles [19], which is well suited to perform operations such as opening or closing required in the real-time application described in Section I. An example of ε with a decomposed SE is shown in Fig. 5, were the SE is decomposed into SE_1 and SE_2, see (5). The input is first eroded by SE_1 and then by SE_2.

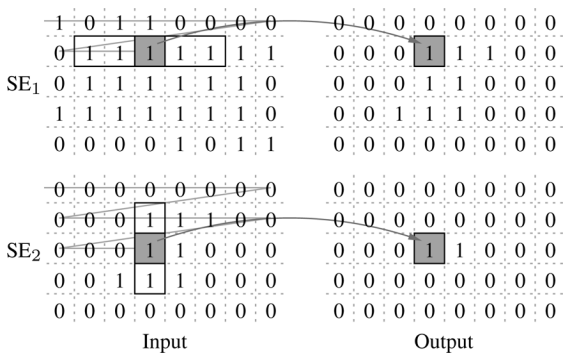

Fig. 5. Input and output of an erosion where a SE of size 3×5 is decomposed into $\mathrm{SE}_1 = 1 \times 5$ and $\mathrm{SE}_2 = 3 \times 1$

Using a flat rectangular SE, ε can be performed as a summation followed by a comparison. Erosion is performed by keeping track of the bits in I that are currently covered by the SE and are compared to its size. Decomposing the SE, the summation can be broken up into two stages. The first stage compares the number of consecutive ONES in I to the width of SE_1. The second stage sums the result from the first stage for each column and compares it to the height of SE_2. If both these conditions are fulfilled, the output at the coordinate of the SE origin is set to ONE, else ZERO.

The proposed architecture is based on the observations above and is shown in Fig. 6 with corresponding word-length (WL) in each stage. Taking advantage of the duality property, the same inner kernel is used for both δ and ε; to perform dilation on an erosion-unit, simply invert the input I and the output. This function is performed in Stage-0 and 3.

To handle the boundary discussed in Section II, the padding is split into four parts: north, east, south, and west, illustrated in Fig. 2(b). Assuming a centered origin, the east and west padding extend $\lfloor \mathrm{SE}_{\text{width}}/2 \rfloor$ columns and the north and south padding extend $\lfloor \mathrm{SE}_{\text{height}}/2 \rfloor$ rows outside I, where $\mathrm{SE}_{\text{height}}$ and $\mathrm{SE}_{\text{width}}$ corresponds to the height and width of the SE, respectively. Since a δ is transformed into an ε by inverting the input (I'), the padding will be ONES regardless of the executed operation. In the proposed architecture, the padding to the west and north are pre-calculated values and inserted as initial values of the sums in stage-1 and 2, respectively. The east and south padding are handled differently since it has to be inserted as extra pixels in the data stream; the east padding in between the rows of I and the south padding after the last pixel have been processed from I. Fig. 2(b) shows an example of both the pre-calculated padding and the inserted extra bits for each corresponding side of I when the SE consists of seven rows and five columns of ONES.

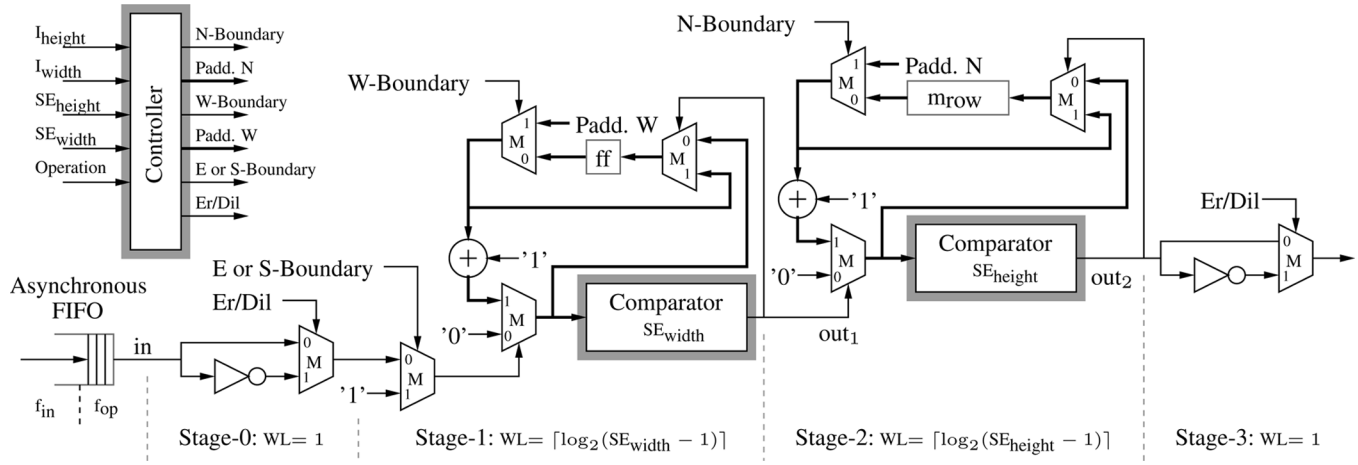

Fig. 6. Architecture of the erosion and dilation unit. Multiplexors are marked with M, the flip-flop with ff and the row memory with $\mathrm{m_{row}}$. I_{height} and I_{width} corresponds to the height and width of the input image and thick lines indicate buses with corresponding WL shown in each stage.

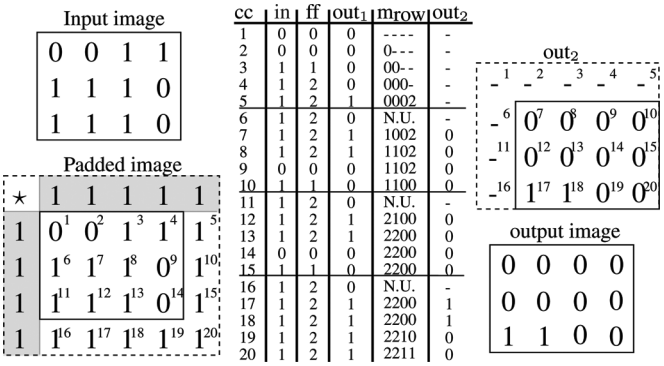

Fig. 7. Clock cycle (cc) true example of an erosion using a 3×3 SE. ff shows the content of the flip-flop in stage-1, $\mathrm{m_{row}}$ the row memory in stage-2, and out_1 and out_2 show the output from respective stage. $N.U.$ indicates that the row memory is not updated in that clock cycle and '$-$' represents invalid data.

With this architecture each pixel in I is used once to update the sum stored in the flip-flop in stage-1, that records the number of consecutive ONES to the left of the currently processed pixel. When the input is ONE, the sum is increased, else reset to ZERO. Each time the sum plus the input equals the width of SE_1, stage-1 outputs a ONE to stage-2 and the previous sum is kept. The same principle is used in stage-2 but instead of a flip-flop, a row memory is used to store the number of ONES from stage-1 in the vertical direction for each column in I. In addition, a controller is required to handle padding and to determine the operation to be performed, i.e., ε or δ. An example of the values in the main blocks in the architecture after each clock cycle when performing an erosion is shown in Fig. 7. The input image is padded in the same manner as shown in Fig. 2(b) and all signals can be found in Fig. 6. Since an erosion is performed, stage-0 and stage-3 are only bypassing the input and output signals.

The input and output of this architecture is binary and hence the WL in stage-0 and stage-3 only has to be one bit. However, in stage-1 and stage-2 sums are recorded and the WL has to be wide enough to hold the maximum values. In stage-1 the maximum sum is equal to $\mathrm{SE_{width}} - 1$ and the corresponding WL to $\lceil \log_2(\mathrm{SE_{width}} - 1) \rceil$. In stage-2 the maximum sum depends on height of the SE and the $\mathrm{WL} = \lceil \log_2(\mathrm{SE_{height}} - 1) \rceil$.

That padding is inserted into the data stream means that input data have to be stalled for the duration of the padding. The effect of this is twofold; additional memory is required and operating frequency of the data-path has to be higher than the input frequency. Hence, an asynchronous FIFO, located at the input in Fig. 6, is needed to store input data and separates the two different clock domains.

C. Stall-Free Low-Complexity Architecture

In order to improve memory requirements, an extension to the architecture described in Section III-B is proposed. The result is an architecture that shares the same principles and explores the same morphological properties, achieving the same computational complexity. However, the major difference lies in how the padding is addressed. Adding hardware support for processing padding in parallel instead of in serial, e.g., the east and west padding, omits the need to stall the input. Hence, no FIFO is required at the input and the memory requirements are reduced even further.

Assuming that the input is streaming back-to-back images, two cases of independent consecutive pixels can be recognized: the transition from one row to the next and the transition from one image to the next. In the row-to-row case, the last pixels in a row are predefined padding pixels and the first pixels in the next row can only increase the stored sum in stage-1 but not produce an output to stage-2. Hence, a modified version of stage-1, which only handles the padding, can be added to process the last pixels in a row corresponding to the east padding, thus freeing the regular stage-1 to start processing the first input pixels of the next row concurrently. The procedure is illustrated in Fig. 8 which shows an example of which padding pixels that are processed in parallel during a transition between two consecutive rows in a frame at time t and $t+1$. Fig. 9(b) shows the dataflow in the stall-free architecture during a row-break. In the image to image case the same principle can be used, since the last pixels in an image are the south padding and the first pixels of the next

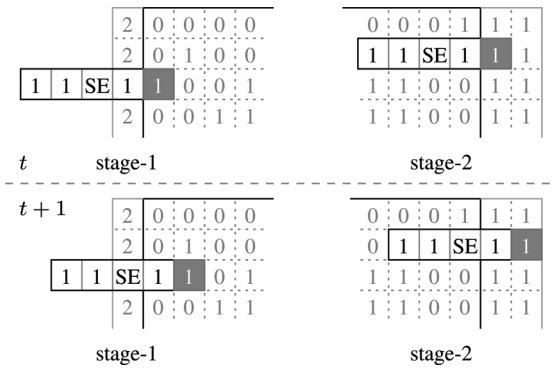

Fig. 8. Example of parallel processing of the padding in stage-1 and 2 at time t and $t+1$ if the SE is of size 1×5. The two east padding pixels from the previous row are processed in parallel with the first two pixels in the current row.

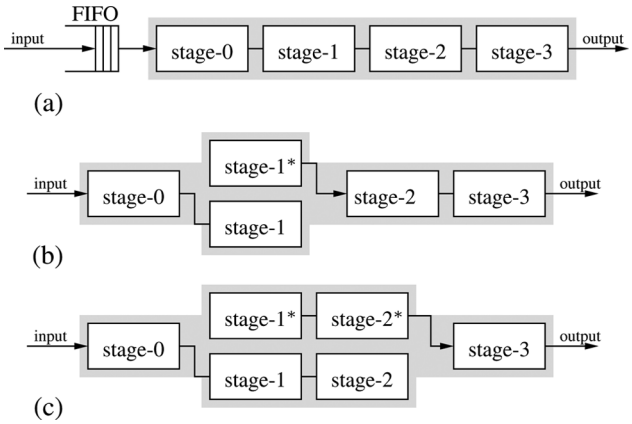

Fig. 9. (a) Block diagram of the low-complexity algorithm. (b), (c) Block diagrams of the dataflow during a row-break and an image-break, respectively, in the stall-free algorithm. $*$ indicates modified blocks.

image cannot produce an output from stage-2. The only difference is that a second stage-2 is added and the dataflow is as shown in Fig. 9(c).

A continuous data stream constituting of back-to-back images is a worst-case scenario assumption. This type of input pattern characteristics can be found when the input source is burst read from a memory, e.g., when processing images in a video sequence. However, when using a sensor in the image acquisition step in a real-time environment, the timing model can be somewhat relaxed. This is due to that for most sensors, there are typically tens of extra cycles in between rows and images in the sensor output pattern. These extra cycles can be utilized to perform the east padding found in between rows and thereby reduce the size requirement of input FIFO needed in the low-complexity architecture. However, the number of extra cycles in the output pattern will in most cases not exceed the number of the required stall cycles during the south padding, which is mainly proportional to $\lfloor \mathrm{SE}_{\mathrm{height}}/2 \rfloor \cdot I_{\mathrm{width}}$, where I_{width} is the width of the input image. Therefore, in applications where the input source is a stream of multiple images requiring an $\mathrm{SE}_{\mathrm{height}} > 1$, the FIFO is still needed in the low-complexity architecture making the stall-free architecture superior in terms of memory requirements compared to the others.

With these modifications, streaming back-to-back images can be processed without stalling input data. Even though the amount of hardware is increased inside the data-path, it is shown in Section V that this amount is far less than the FIFO requirement. Another benefit derived from this property is that only one clock domain is required, i.e., the architecture can run at the same speed as the incoming data, which facilitates incorporating the unit in an embedded system environment.

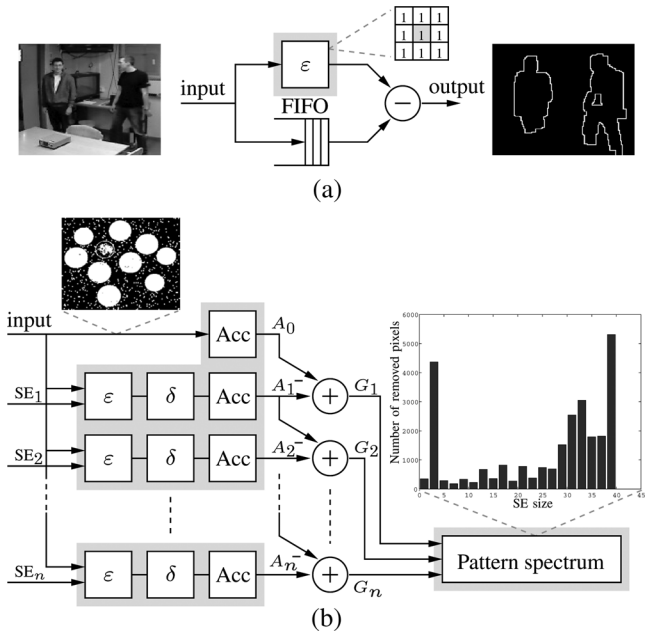

Fig. 10. Examples of extended morphological operations based on the low-complexity architecture. (a) Boundary extraction shown together with the required SE with shaded origin and result. (b) Hardware unit for pattern spectrum extraction. The pattern spectrum has mainly three peaks indicating the size of the clusters, i.e., approximately $3 \times 3 = 9$, $33 \times 33 = 1089$ and $39 \times 39 = 1521$ pixels.

D. Extended Morphological Operations

Due to its low-complexity, the stall-free architecture allows several units to be connected to form extended morphological operations, which increases the flexibility and thereby the applicability of the architecture. As an example, contour extraction is performed by subtracting $I - \varepsilon(I, \mathrm{SE} = 3 \times 3)$, which is accomplished with an adder and a FIFO to compensate for the latency imposed by the architecture [20]. A boundary extraction unit using the proposed architecture together with examples of input and output is shown in Fig. 10(a).

Granulometry based on parallel openings is a morphological operation which is used to estimate cluster sizes in images [21]. This is an example of an advanced operation in which the benefits of the proposed architecture are substantial both in terms of speed and memory requirements. The operation is based on the difference between the remaining number of pixels after parallel openings with an increasing SE size, i.e., a square with a side

$N \in \{1,3,5,\ldots,n\}$. Let A_i be the sum of the remaining pixels p equal to ONE after the ith opening, calculated as

$$A_i = \sum_{\forall p=1} I \circ \mathrm{SE}_i$$

referred to as the size distribution. The difference between adjacent size distributions is defined as the granulometric function and is calculated according to

$$G_i = A_{i-1} - A_i, \qquad \text{where } i = \{1,2,\ldots,n\}$$

which is also denoted pattern spectrum. The granulometric function is sensitive to changes in the number of removed pixels which means that an impulse in the pattern spectrum at a certain SE size indicate that this is a typical cluster size. A hardware unit to calculate the pattern spectrum based on the proposed architecture is illustrated in Fig. 10(b), where Acc are accumulators that sums and store the number of remaining foreground pixels. When all openings are finished, G_1 to G_n are calculated as the difference between the sums A_0 to A_n. To determine the number of opening branches n, some *a priori* knowledge of the image content is required. This is application specific and depends on the relation between the SE size and the resolution. However, even with a large number of branches, e.g., a quarter of the image height, the memory requirement in the unit is still low due to the use of the proposed architecture, i.e., mainly proportional to $I_{\mathrm{height}}/4(\lceil \log_2(\mathrm{SE}_{\mathrm{height}}) \rceil I_{\mathrm{width}})$, where I_{height} is the height of the input image. The unit preserves the important property of streaming data and can run at the same speed as the incoming pixels, but with a latency proportional to the largest SE size. Examples of where granulometry can be useful is process monitoring and in medical applications [22].

To be able to perform other important and more computationally expensive multi-pass morphological operations such as the hit-and-miss, skeletonization, and convex hull transformation [23], additional intermediate storage as well as an extension to the supported SE is required.

IV. Implementation

The architectures have been implemented in VHDL and synthesized for the UMC 0.13-μm CMOS process, supporting an image resolution of 640 $\times$ 480 and a maximum SE size of 63 $\times$ 63 pixels (not limited by the architecture). All three architectures can perform either ε or δ, controlled by a single bit, and support changing the SE size during run-time, i.e., height and width. Table I compiles the most important resource requirements and characteristics of the architectures. Memory area is divided into $\mathrm{mem}_{\mathrm{fifo}}$ and $\mathrm{mem}_{\mathrm{dp}}$, where the former is the amount of memory used to stall or align input data and the latter is the required memory to calculate the output.

Table I shows that memory is a significant part of all three implementations. The delay-line architecture has a lower memory area percentage than the other two since this architecture has a more complex controller which handles the padding. For both the low-complexity and the stall-free architecture, memory is

TABLE I
Synthesis Results in the UMC 0.13-μm CMOS Process, Using an Image Resolution of 640 $\times$ 480 and Supporting a Maximum SE Size of 63 $\times$ 63 Pixels

Design	Delay-line	Low-complexity	Stall-free
Memory [KB]	4.97	3.08	0.96
$\mathrm{mem}_{\mathrm{fifo}}$ area [mm^2]	0.25	0.23	0
$\mathrm{mem}_{\mathrm{dp}}$ area [mm^2]	0.13	0.05	0.10
Total area [mm^2]	0.88	0.29	0.11
Memory area	43%	98%	88%
Normalized area	7.7	2.5	1
Gate count [k]	172	56	22
Max speed [MHz]	333	190	333

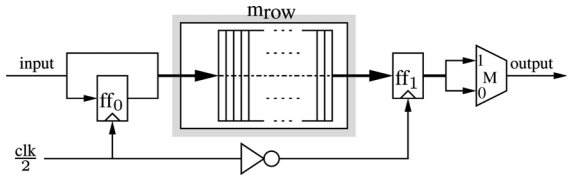

Fig. 11. Row memory implemented with one double-width single-port memory, which performs read and write every other clock cycle.

equal to or more than 88% of the total area when the row memories are implemented as single-port high-density SRAMs, further discussed in Section IV-A. In order to distinguish the area requirement relationship between the designs, the normalized area is inferred. This figure shows that the delay-line and low-complexity architecture requires a factor of 7.7 and 2.5 more area than the stall-free architecture. Furthermore, it can be noticed that the low-complexity architecture has a reduced operating speed compared to the others. This is due to that the asynchronous FIFO located at the input of this architecture is replaced by a dual-port memory. The gate count is based on a 2-input NAND-gate (5.12 μm^2) and includes all memory blocks.

A. Memory Architecture

In the low-complexity and stall-free architecture, memory is by far the single largest component putting constraints on maximum operating speed as well as a lower limit on the area. Therefore, it is of special interest to optimize the memory requirements. Ideally, a value in the row memory $(\mathrm{m}_{\mathrm{row}})$ should be read, updated, and written back to the memory in a single cycle. This requires a simultaneous read and write operation that normally is implemented using a dual-port memory. However, this type of memory introduces an area overhead mainly due to the dual address decoders, especially large if the memory is small. Another observation is that the row memories have a memory content access pattern of a FIFO, resulting in that the address generation becomes trivial and can be implemented as a simple modulo-counter. Based on these facts, all row memories can be advantageously implemented using a single-port memory of double width that reads and writes two samples every other clock cycle. The architecture is illustrated in Fig. 11. As an example, using a memory with a depth and width of 320 $\times$ 12 bits and two 12 bit flip-flops, the memory area can be reduced

by approximately 30% compared to using a standard dual-port memory for this particular process (UMC 0.13-μm).

V. Results and Performance

This section discusses and compares the performance of each architecture. The comparison is performed in terms of computational complexity, execution time, and memory requirements.

A. Computational Complexity

Computational complexity for the presented architectures is measured in number of operations per output, i.e., the number of times an input sample is used. Typically, the delay-line architecture uses each input as many times as the number of elements in the SE in order to support arbitrary shaped SEs, but can be reduced to $2\lceil\log(N_{\rm se})\rceil$ when using a rectangular SE (discussed in Section I-A). Both the low-complexity and the stall-free architecture have a constant computational complexity of 4 operations per pixel, i.e., each operation is accomplished with only two summations and two comparisons and use each input only once, independent of the SE size. This is due to that they are based on the same principle which is to trade the freedom of choosing an arbitrary SE shape for reduced complexity.

B. Execution Time

In a typical morphological operation, the execution time $T_{\rm exe}$ is the time between processing the first input until the last output has been produced. It consists of two contributions: pixel processing time, $T_{\rm pp}$, and padding time, $T_{\rm padd}$. $T_{\rm pp}$ is the time it takes for the architecture to process all the pixels in the input image and is thus proportional to the resolution and in some cases the SE size, depending on if time multiplexing is used in the implementation. $T_{\rm padd}$ includes all extra clock cycles due to padding and is hence dependent on both the resolution and SE size, as shown in Fig. 2.

The execution time, measured in clock cycles, of the delay-line and stall-free architecture is equal to the image resolution, since no padding is inserted into the data stream, and can be written as

$$T_{\rm exe} = T_{\rm pp} = I_{\rm height} \cdot I_{\rm width} \text{ clock cycles.} \tag{7}$$

The low-complexity architecture, on the other hand, needs to insert padding on two sides, resulting in an execution time of

$$\begin{aligned} T_{\rm exe} &= T_{\rm pp} + T_{\rm padd} \\ &= (I_{\rm height} \cdot I_{\rm width}) + \left\lfloor \frac{\rm SE_{width}}{2} \right\rfloor I_{\rm height} + \left\lfloor \frac{\rm SE_{height}}{2} \right\rfloor \\ &\quad \times \left(I_{\rm width} + \left\lfloor \frac{\rm SE_{width}}{2} \right\rfloor \right) \text{clock cycles} \end{aligned} \tag{8}$$

where the second and third term corresponds to the time it takes to insert the east and south padding.

Comparing (7) and (8) for an input image of 640×480 and a $\rm SE = 63 \times 63$ it is found that the low-complexity architecture requires approximately 11% longer execution time due to the inserted padding. With an increasing resolution compared to the SE size, this penalty will become smaller and eventually insignificant. However, this architecture still requires multiple clock domains; one for the pixel stream and one for the operating frequency of the architecture.

C. Memory Requirement

The required amount of memory for the delay-line architecture can be seen in Fig. 3(b) and is calculated as

$$\begin{aligned} \text{mem}_{\rm dl} = ({\rm SE_{height}} - 1)(I_{\rm width} - {\rm SE_{width}} + 1) \\ + {\rm SE_{height}}({\rm SE_{width}} - 1) \text{bits} \end{aligned} \tag{9}$$

where the first term accounts for the FIFOs and the second term for the flip-flops used to extract the SE.

The memory requirement for the low-complexity architecture is proportional to the word-length in each stage, illustrated in Fig. 6. The word-lengths in stage-1 and 2 depend on the maximum supported SE size and is equal to $\lceil\log_2({\rm SE_{width}} - 1)\rceil$ and $\lceil\log_2({\rm SE_{height}} - 1)\rceil$, respectively. In addition, a FIFO is required at the input since the incoming pixel stream needs to be stalled during the processing of the padding pixels. Thus, the total amount of required memory is

$$\text{mem}_{lc} = \text{FIFO} + \lceil\log_2({\rm SE_{width}})\rceil + \lceil\log_2({\rm SE_{height}})\rceil I_{\rm width} \text{bits} \tag{10}$$

where the second and third term corresponds to the flip-flop in stage-1 and to the row memory in stage-2, respectively. The size of the FIFO not only depends on the padding and resolution but also on the operating and input frequency, $f_{\rm op}$ and $f_{\rm in}$; the higher $f_{\rm op}$ compared to $f_{\rm in}$, the smaller the FIFO. If $f_{\rm op}$ lowered as much as possible while still supporting back-to-back images, i.e., $f_{\rm op} = ((N + N_p)/N) f_{\rm in}$, the size of the FIFO can be approximated as

$$\text{FIFO} \approx N_{\rm sp} \frac{f_{\rm in}}{f_{\rm op}} = N_{\rm sp} \frac{N}{N + N_p} \approx N_{\rm sp} \text{bits} \tag{11}$$

where $N_{\rm sp}$, $N_{\rm p}$, and N are the size of the south padding, all padding and the input image, respectively.

The memory requirements of the stall-free architecture follows the principles of (10) but without the FIFO. The resulting total memory requirement can be written as

$$\text{mem}_{\rm sf} = 2\left(\lceil\log_2({\rm SE_{width}})\rceil + \lceil\log_2({\rm SE_{height}})\rceil I_{\rm width}\right) \text{bits} \tag{12}$$

where the factor 2 is due to the parallel processing during padding. However, removing the FIFO at the input still has a significant impact on the memory requirements, as shown in Table I. Equation (12) indicates that the memory area of the data-path in the stall-free architecture should be twice the size of the low-complexity memory, but this is not the case when comparing $\text{mem}_{\rm dp}$ in Table I. The explanation is that instead of using two separate row memories, one single row memory of double width is used as discussed in Section IV-A.

Hardware requirements for implementations supporting higher resolutions and SEs can be estimated accurately by using the required memory size since this is the main source, as shown in Table I. The memory requirements for the three architectures as a function of SE size using an image resolution of 1280×1024 is shown in Fig. 12. As an example, the total memory requirement of a stall-free architecture supporting

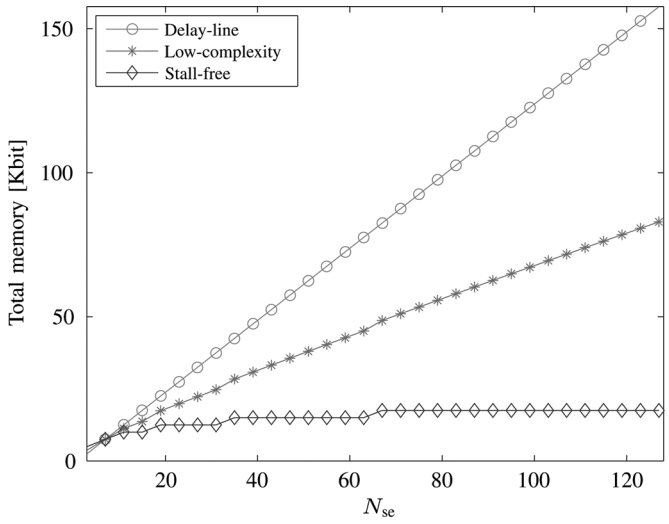

Fig. 12. Vertical axis shows the total memory requirement in kbit as a function of N_{se} for each implementation. The image resolution is 1280 × 1024 and N_{se}^2 is the size in pixels of a quadratic SE.

TABLE II
THE MOST IMPORTANT PROPERTIES OF THE ARCHITECTURES, WHERE N^2 AND N_{SE}^2 IS THE SIZE IN PIXELS OF A QUADRATIC INPUT IMAGE AND SE

Design	Delay-line	Low-complexity	Stall-free
Complexity	$2\lceil \log_2(N_{\mathrm{se}}) \rceil$	4	4
mem accesses	$2N_{\mathrm{se}}$	$2+\alpha$	$2+\alpha$
mem [bits]	$N\,N_{\mathrm{se}}$	$N\,\log_2(N_{\mathrm{se}})$	$N\,\log_2(N_{\mathrm{se}})$
$\mathbf{T}_{exe}$	N^2	N^2+$N_{\mathrm{se}}\,N$	N^2
SE support	Arbitrary	Rectangular	Rectangular

a maximum SE size of 63 × 63 is about 15 kbits. With the same settings, the delay-line implementations would require approximately 79 kbits of memory, which is more than 5 times as much. Table II summarizes the most important properties of the different architectures as functions of image resolution and SE size. The table clearly indicate that for applications in which a rectangular SE is sufficient to fulfill the specifications, the stall-free architecture reduces computational complexity and memory requirements without sacrificing execution time.

From a power perspective it is advantageous to have a low arithmetic complexity and to limit the number of memory access since both contribute to the dynamic power budget. The actual number of memory accesses per pixel for the delay-line implementation is mainly proportional to $2 \cdot (\mathrm{SE}_{\mathrm{height}} - 1)$, since each value is shifted downward one row in Fig. 3 each time it is to be used in a calculation (the factor two is for reading and writing). For the low-complexity and the stall-free architecture, this number is reduced to $2 + \alpha$, where α is additional $2\lfloor \mathrm{SE}_{\mathrm{height}}/2 \rfloor (I_{\mathrm{width}})$ accesses required during the south padding (neglecting the input FIFO read and and write operation required in the low-complexity architecture). As an example, using a resolution of 640 × 480 and supporting a maximum SE size of 63 × 63, these additional memory accesses only constitutes ≈6.5% of the total number memory access per frame or additional $\alpha = 0.13$ accesses per pixel. The conclusion is that both the low-complexity and the stall-free architecture mainly only require one read and one write operation per pixel. Furthermore, since static power consumption is becoming increasingly important in modern CMOS technologies due to leakage, it is beneficial to reduce the overall area [24]. For the designs in this article, area mainly constitutes of memory. A reduced memory requirement will therefore not only have a large impact on the static but also on the dynamic power since accessing smaller memories requires less power than larger ones. Based on these facts and the results in Table II, it is seen that the stall-free low-complexity architecture has the lowest complexity and lowest memory requirements both in terms of bits and accesses, hence has better dynamic and static power dissipation properties than the other designs.

VI. CONCLUSION

This article presents an evaluation of three architectures for binary erosion and dilation intended to be used as hardware accelerator in real-time applications. In particular, an architecture of a fast stall-free low-complexity architecture based on SE decomposition is proposed. The most important features and properties are that it requires no extra clock cycles due to padding and has a memory requirement proportional to the SE height and input image width. The architecture supports flat arbitrary sized rectangular SEs and the number of operations and memory accesses per pixel is constant, independent of both the SE and image size. Furthermore, due to its low complexity and memory requirement, multiple units can be connected without any intermediate storage to perform other morphological operations. In order to verify and evaluate the results, the architecture has been implemented in VHDL and synthesized in the UMC 0.13-μm CMOS process using a resolution of 640 × 480 and supporting a maximum SE of 63 × 63. In comparison with implementations of the delay-line and the low-complexity architecture using the same parameter setting, the area is decreased by a factor of 7.7 and 2.5, respectively, while achieving the same or better execution time.

ACKNOWLEDGMENT

The authors would like to thank E. Ledfelt, Ericsson Mobile Platforms for valuable input to this work.

REFERENCES

[1] J. Serra, *Image Analysis and Mathematical Morpohology*. New York: Academic Press, 1982.
[2] E. R. Dougherty and R. A. Lotufo, *Hands-on Morphological Image Processing*. Bellingham, WA: Spie Press, 2003.
[3] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Comput. Soc. Conf. Computer Vision Pattern Recogn.*, Ft. Collins, TX, Jun. 23–25, 1999.
[4] B. Kisačanin and D. Schonfeld, "A fast thresholded linear convolution representation of morphological operations," *IEEE Trans. Image Process.*, vol. 3, no. 4, pp. 455–457, Jul. 1994.
[5] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: Implementation and evaluation," *J. Signal Process. Syst.*, vol. 52, no. 1, Jul. 2008.
[6] H. Jiang, H. Ardö, and V. Öwall, "Real-time video segmentation with VGA resolution and memory bandwidth reduction," in *Proc. 2006 IEEE Int. Conf. Advanced Video and Signal based Surveillance*, Sydney, Australia, Nov. 2006.
[7] S. Fejes and F. Vajda, "A data-driven algorithm and systolic architecture for image morphology," in *Proc. IEEE Int. Conf. Image Process.*, Austin, Texas, Nov. 13–16, 1994, vol. 2, pp. 550–554.

[8] J. Velten and A. Kummert, "FPGA-based implementation of variable sized structuring elements for 2-D binary morphological operations," in *Proc. 1st IEEE Int. Workshop Electron. Design, Test, Appl.*, Jan. 29–31, 2002, pp. 309–312.
[9] S. Y. Chien, S. Y. Ma, and L. G. Chen, "Partial-result-reuse architecture and its design technique for morphological operations with flat structuring element," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 344–371, Sep. 2005.
[10] A. Źarandy, A. Stoffels, T. Roska, and L. O. Chua, "Implementation of binary and gray-scale mathematical morphology on the cnn universal machine," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 45, no. 2, pp. 163–168, Feb. 1998.
[11] E. N. Malamas, A. G. Malamos, and T. A. Varvarigou, "Fast implementation of binary morphological operations on hardware-efficient systolic architectures," *J. VLSI Signal Process.*, vol. 25, pp. 79–93, 2000.
[12] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Pattern Recogn. Lett.*, vol. 13, no. 7, pp. 517–521, 1992.
[13] H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "A low complexity architecture for binary image erosion and dilation using structuring element decomposition," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kobe, Japan, May 2005, vol. 4, pp. 3431–3434.
[14] J. Goutsias and H. J. Heijmans, "Fundamenta morphologicae mathematicae," *Fund. Info.*, vol. 41, no. 1–2, pp. 1–31, Jan. 2000.
[15] J. Velten and A. Kummert, "Implementation of a high-performance hardware architecture for binary morphological image processing operations," in *Proc. IEEE Int. Midwest Symp.Circuits Syst.*, Hiroshima, Japan, Jul. 25–28, 2004, vol. 2, pp. 241–244.
[16] H. Park and R. T. Chin, "Decomposition of arbitrarily shaped morphological structuring elements," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, no. 1, pp. 2–15, Jan. 1995.
[17] G. Anelli and A. Broggi, "Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, no. 2, pp. 217–224, 1998.
[18] F. Y. Shih and Y. T. Wu, "Decomposition of binary morphological structuring elements based on genetic algorithms," *J. Comput. Vision Image Understand.*, vol. 99, no. 2, pp. 291–302, 2005.
[19] R. Lam and C. Li, "A fast algorithm to morphological operations with flat structuring element," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 45, no. 3, pp. 387–391, Mar. 1998.
[20] R. Gonzalez and R. Woods, *Digital Image Processing*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.
[21] G. Matheron, *Random Sets and Integral Geometry*. New York: Wiley, 1975.
[22] A. G. Dempster and C. D. Ruberto, "Using granulometries in processing images of malarial blood," in *Proc. IEEE Int. Symp. Circuits Syst.*, Sydney, Australia, May 2001.
[23] P. Soille, "From binary to gray scale convex hulls," *Fund. Inf.*, vol. 41, pp. 131–146, Jan. 2000.
[24] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuit*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.

Hugo Hedberg received the M.S.E.E. and Ph.D. degrees in electrical engineering from Lund University, Lund, Sweden, in 2001 and 2008, respectively.

His doctoral thesis addresses hardware accelerators for automated digital surveillance systems. His main research area is hardware implementations of image processing algorithms targeted for real-time embedded systems with a special interest in developing low-complexity architectures for morphological operations. He is currently with Prevas, Stockholm, Sweden.

Fredrik Kristensen received the M.S.E.E and Ph.D. degrees in electrical engineering from Lund Institute of Technology, Lund University, Lund, Sweden, in August 2001 and September 2007, respectively.

His doctoral thesis related to hardware implementations of embedded automated surveillance systems. His main research area is real-time video processing and is currently with Nokia, Copenhagen, Denmark, working as a Hardware Designer.

Viktor Öwall received the M.Sc. and Ph.D. degrees in electrical engineering from Lund University, Lund, Sweden, in 1988 and 1994, respectively.

During 1995 to 1996, he joined the Electrical Engineering Department, the University of California at Los Angeles as a Postdoctoral Researcher, where he mainly worked in the field of multimedia simulations. Since 1996, he has been with the Department of Electrical and Information Technology, Lund University. His main research interest is in the field of digital hardware implementation, especially algorithms and architectures for wireless communication, image processing and biomedical applications. Current research projects include combining theoretical research with hardware implementation aspects in the areas of pacemakers, channel coding, video processing, and digital holography.

Dr. Öwall was Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING from 2000–2002 and is currently Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.