

Research Article

Resource Efficient Hardware Architecture for Fast Computation of Running Max/Min Filters

Cesar Torres-Huitzil

Information Technology Laboratory, CINVESTAV, Km. 5.5 Carretera Ciudad Victoria-Soto La Marina, 87130 Ciudad Victoria, TAMPS, Mexico

Correspondence should be addressed to Cesar Torres-Huitzil; ctorres@tamps.cinvestav.mx

Received 24 August 2013; Accepted 17 September 2013

Academic Editors: B. Sun and J. Zhang

Copyright © 2013 Cesar Torres-Huitzil. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Running max/min filters on rectangular kernels are widely used in many digital signal and image processing applications. Filtering with a $k \times k$ kernel requires of $k^2 - 1$ comparisons per sample for a direct implementation; thus, performance scales expensively with the kernel size k . Faster computations can be achieved by kernel decomposition and using constant time one-dimensional algorithms on custom hardware. This paper presents a hardware architecture for real-time computation of running max/min filters based on the van Herk/Gil-Werman (HGW) algorithm. The proposed architecture design uses less computation and memory resources than previously reported architectures when targeted to Field Programmable Gate Array (FPGA) devices. Implementation results show that the architecture is able to compute max/min filters, on 1024×1024 images with up to 255×255 kernels, in around 8.4 milliseconds, 120 frames per second, at a clock frequency of 250 MHz. The implementation is highly scalable for the kernel size with good performance/area tradeoff suitable for embedded applications. The applicability of the architecture is shown for local adaptive image thresholding.

1. Introduction

Running max/min filtering is an important operation that aims at selecting the maximum or minimum value from a set of signal elements. A window moves over all data items and at each point the max/min value of the data within the window is taken as output [1]. Max/min filters are widely used in tasks such as noise filtering, adaptive control, pattern recognition, and speech and image processing. The max filter, in gray-level image morphology [2], corresponds to the dilation operator over images using a flat structuring element (SE) or kernel, and the min filter corresponds to the erosion operator. These filters are very attractive since their computation requires only comparisons and no other arithmetic operations and because of their robust behavior in the presence of noise and signal nonstationarities [3–5].

For some image-based industrial applications, such as granulometries, particle size distribution, or local adaptive binarization, the filtering of high resolution images with large two-dimensional kernels could be very time consuming. A

direct evaluation of such filters leads to $O(k^2)$ comparisons per sample, being k the size of the kernel. A possible alternative to speed up computations is to decompose large kernels into linear or simpler ones [6]. Then, linear filtering might be implemented by efficient algorithms [7] and/or by dedicated hardware structures. Under this approach, the HGW algorithm is a widely used method to compute max/min with linear kernels whose complexity is independent of the filter size [8, 9]. Motivated by the advantages of kernel decomposition, the existence of efficient one-dimensional filtering algorithms, and by the challenge to handle the computational cost and memory requirements, the design of an architecture for fast computation of running max/min image filters with arbitrary-length rectangular kernels is presented herein. This paper proposes an efficient coarse-grain pipelined implementation of the HGW algorithm as a building block with a memory usage improvement based on distributed memory available on FPGAs compared to previous architectures that use dedicated embedded Block Ram memory.

Running max/min filters have been realized in different implementation media such as very large scale integration (VLSI) circuits and FPGAs. Most of these hardware implementations target rather small rectangular kernels and employ a pipeline technique in which a raster-scan image is sequentially fed into a long delay line and then into an array of neighboring processing elements (PEs) and the max/min operations are carried out in parallel [10, 11]. The strength of such architectures is that they can be pipelined down to a single compare-swap stage yielding high throughput and frequency. Techniques to decrease the number of comparators required to support large SEs were further introduced. For instance, in [12], authors propose a partial-result-reuse (PRR) architecture for gray-level morphological operations with flat SEs. Partial results generated during computations are kept and reused in this architecture to reduce hardware cost. However, a considerable cost, for computation and storage, is still needed for large kernels.

In [13], authors present an efficient hardware architecture to achieve erosion/dilation with very large linear kernels based on a slightly modified HGW algorithm. They propose a block mirroring scheme to suppress the need for backward scanning, to ease data propagation and memory access and to minimize memory consumption. However, embedded memories represent a large part of their design. When synthesized on a Virtex 4LX60 device, the architecture uses 3 Block Rams of 18 Kbits and 700 slices. The maximum kernel size that could be supported is 1023 over a line length of 65535 pixels. The design memory consumption is image size independent, but increasing further the parallelism, for instance, to process several image lines concurrently, is limited due to the number of Block Rams available on FPGAs. In [14], another implementation of erosion/dilation based on SE decomposition and/or efficient 1-D algorithms is proposed. The method is based on a recursive morphological decomposition of 8-convex SEs by using only causal two-pixel SEs. The proposed architecture is generic and fully regular, built from elementary interconnected modules. It has been synthesized into an FPGA, achieving high operation frequencies for any shape and size of SE; however, for large SEs a long pipeline is required.

Although some architectures for max/min filters have been developed, improvements are still needed for filtering high definition video streams in real time. On one hand, embedded memories represent a considerable cost of previous designs limiting FPGA deployment in embedded environments. On the other hand, the scalability is another concern since the architectures need major modifications when the kernel size increases. This is the primary motivation for the proposed optimized implementation, which relies on some architectural techniques used in [13].

The rest of the paper is organized as follows. The HGW algorithm is presented in Section 2. In Section 3, the proposed architecture is presented in detail as well as the strategy for mapping memory requirements to on-chip resources. Section 4 presents the FPGA implementation, experimental results, and local adaptive thresholding as a case of study. Concluding remarks and future work are presented in Section 5.

2. van Herk/Gil-Werman Algorithm

The one-dimensional version of a running max filter of order k can be formulated as follows. Given an input sequence of size M , $f_0, f_1, f_2, \dots, f_{M-1}$, the response r_i of the filter for $i = 0, \dots, M - 1$ is given by the following equation:

$$r_i = \max_{0 \leq j < k} f_{i+j}. \quad (1)$$

In the actual processing of the sequence, the boundaries usually receive some special treatment, for example, padding, periodic condition, and so forth.

The HGW algorithm consists of three main processing steps [8, 9] as illustrated in Figure 1. First, the input sequence is split into segments of length k , where a propagation in a forward way of the current max value of f , for $x = 0, \dots, M - 1$, is done using the following equation:

$$g(x) = \begin{cases} f(x), & \text{if } x \bmod k = 0, \\ \max(g(x-1), f(x)), & \text{otherwise.} \end{cases} \quad (2)$$

In a second processing step, a propagation in a backward way of the current max value of f is performed using the following equation:

$$h(x) = \begin{cases} f(x), & \text{if } x \bmod (k-1) = 0, \\ \max(h(x-1), f(x)), & \text{otherwise,} \end{cases} \quad (3)$$

for $x = M - 1, \dots, 0$. Note that the values in a given segment are scanned in a reverse order to produce $h(x)$ as opposed to $g(x)$.

In the last processing step, the max (or min) is computed by merging the g and h arrays using the following equation:

$$r(x) = \max\left(g\left(x + \frac{k}{2}\right), h\left(x - \frac{k}{2}\right)\right), \quad x = 0, \dots, M - 1. \quad (4)$$

Equations (2), (3), and (4) each require only a single comparison per array element, that is, three comparisons per sample to filter with a linear kernel of any size. The d -dimensional max/min filter can be computed using a kernel decomposition approach by sequentially applying the one-dimensional filter d times. In the two-dimensional case, only 6 comparisons per pixel are required by applying the one-dimensional HWG consecutively to rows and columns of the input image.

The HGW algorithm is amenable for parallelism and coarse-grain pipelining; however, the large data buffers required to store $g(x)$, $h(x)$ and the pipelined computation of $r(x)$ are identified as the most challenging aspects for a hardware implementation. In this sense, the solution proposed in [13] is not fully adequate for embedded scenarios as the memory resources are substantially high. In this paper, an FPGA-based memory resource efficient architecture that exploits parallelism and pipelining is presented. The goal is to achieve an optimized embedded implementation with a high throughput while reducing the dedicated on-chip Block Ram memory by an efficient utilization of the distributed memory resources available in current FPGA devices.

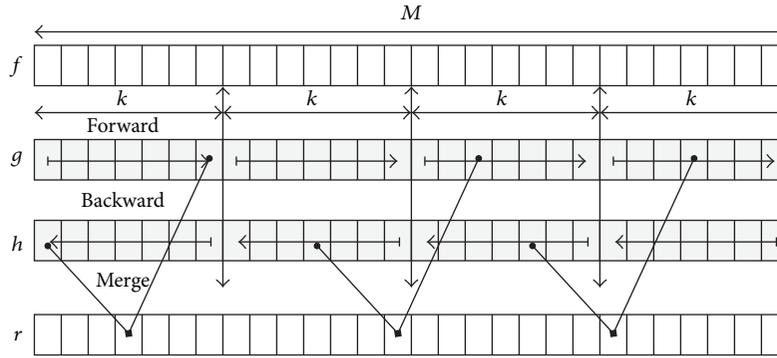


FIGURE 1: A simplified view of the main steps, forward, backward, and merge, of the HGW algorithm for a linear kernel of size k over an input sequence f of size M .

3. Proposed Architecture

The HGW algorithm in addition to modularity and regularity exhibits the following desirable properties for a hardware implementation [1]: (1) operations reducing to few comparator modules, (2) local and regular data and control flow requirements, and (3) inherent pipelining and multiprocessing features. The whole architecture is examined and its main components are described in detail in the following subsections.

3.1. Architecture Overview. A block diagram of the hardware architecture for the HWG algorithm is shown in Figure 2. A set of three comparators are required for the internal computations, which are well mapped to the FPGA resources. The comparators are labeled as forward, backward, and merge to indicate to which processing step each comparator belongs to. The counter-based control unit synchronizes all data and control flow among modules in the architecture. Additionally, it generates the external memory addresses both to read data from the input memory and to store the processed data in the output memory according to image and kernel specifications. For the purpose of simplicity, just the main control signals are shown in Figure 2.

The major building blocks in the proposed architecture are memory units since the HGW algorithm is memory centric in the sense that more resources are required for internal storage than for computation. This is an example of the so-called systems on-chip that requires frequent sharing, communication, queuing, and synchronization among distributed functional units [15].

3.2. Memory Organization and Mapping. The architecture memory organization is based on the scheme used in [13], where three Block Ram modules were used. However, herein a different strategy is used to map logical memories to FPGA distributed memory such that function-level parallelism can be further exploited to improve scalability and performance. In [13], dual-port memories were used to ease the propagation and memory accesses. In addition, a block mirroring strategy was proposed to suppress the need for a complete backward scanning of the input stream. The mirroring scheme requires two Block Rams, RAM_2 and RAM_3 , whose depth depends on

the maximum supported kernel size. The third memory unit, RAM_1 , provides temporal storage for the computed values in the forward step and to synchronize the pipeline stages as will be explained in the following section.

In this paper, we use the memory resources distributed across the FPGA instead of the embedded Block Rams of unique size (i.e., 18 Kbit for Xilinx Spartan-6 devices). Some LUTs within each configurable logic block (CLB) optionally implement a 16×1 -bit synchronous RAM which can be cascaded for deeper and/or wider memories [16]. Distributed RAM writes synchronously and reads asynchronously. This property is exploited in this work so as to avoid the use of dual-port memories. Furthermore, the address port either for single- or dual-port modes is asynchronous with an access time equivalent to a LUT logic delay.

3.3. Architecture Pipeline Scheme. Figure 3 shows a high-level overview of the three-stage pipeline scheme used in the architecture in order to sustain a high output data rate. The internal memory requirements of the pipeline are provided by using low overhead on-chip memory, distributed synchronous single port rams, that alleviates the need of dual-port rams. The three memory units can be implemented very efficiently on FPGAs by taking advantage of concurrent synchronous writing and asynchronous reading since only streaming operations are required on windows of at most k elements.

The coarse-grain computational stages in the pipeline can be described as follows. In the first stage, two processing tasks are performed concurrently on the incoming data stream. The max value is propagated in a forward way and the stream values also undergo a reverse order arrangement in segments of size k . The second pipeline stage starts its operation after k clock cycles, and it performs the forward propagation of the previous mirrored segment and a backward mapping is also applied. The third stage starts the computations after the second stage completes the computation of $k/2$ output samples. Since the merge stage requires the data computed by the forward and backward stages becomes available, its operation must be delayed $3k/2$ clock cycles after it can operate continuously on the $g(x)$ and $h(x)$ streams as shown in Figure 3. For synchronization purposes, the values of the forward stage must be delayed k clock cycles. This buffering is

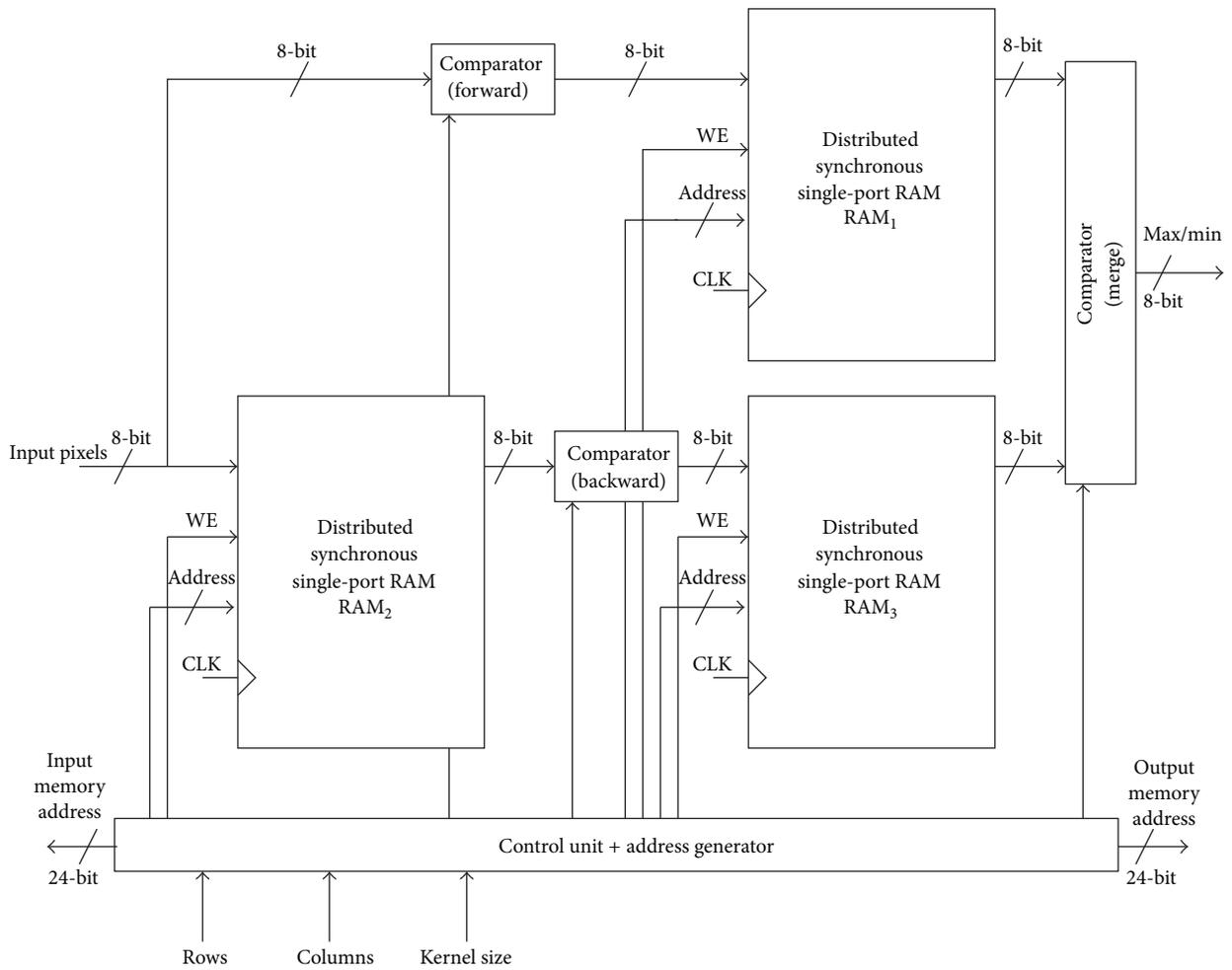


FIGURE 2: Block diagram of the proposed hardware architecture for running max/min filters based on the HWG algorithm and its main components.

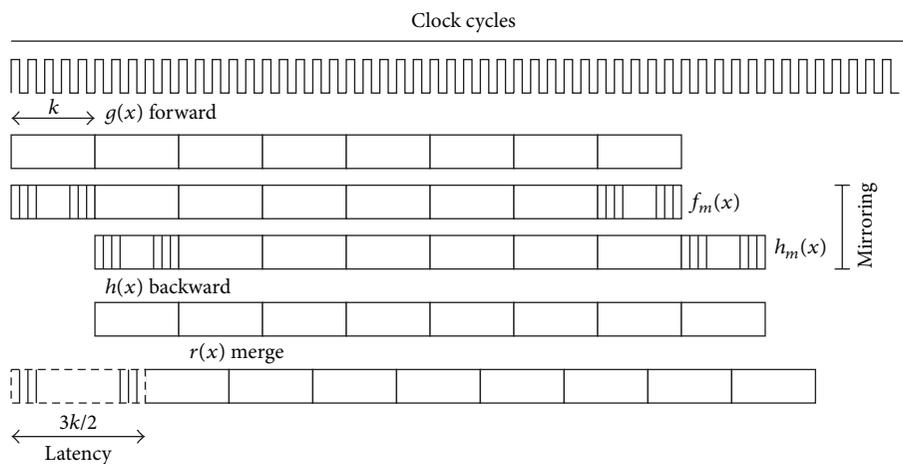


FIGURE 3: Pipeline scheme used in the HGW architecture. The computations and data flow are organized around a three-stage pipeline, forward, backward, and merge.

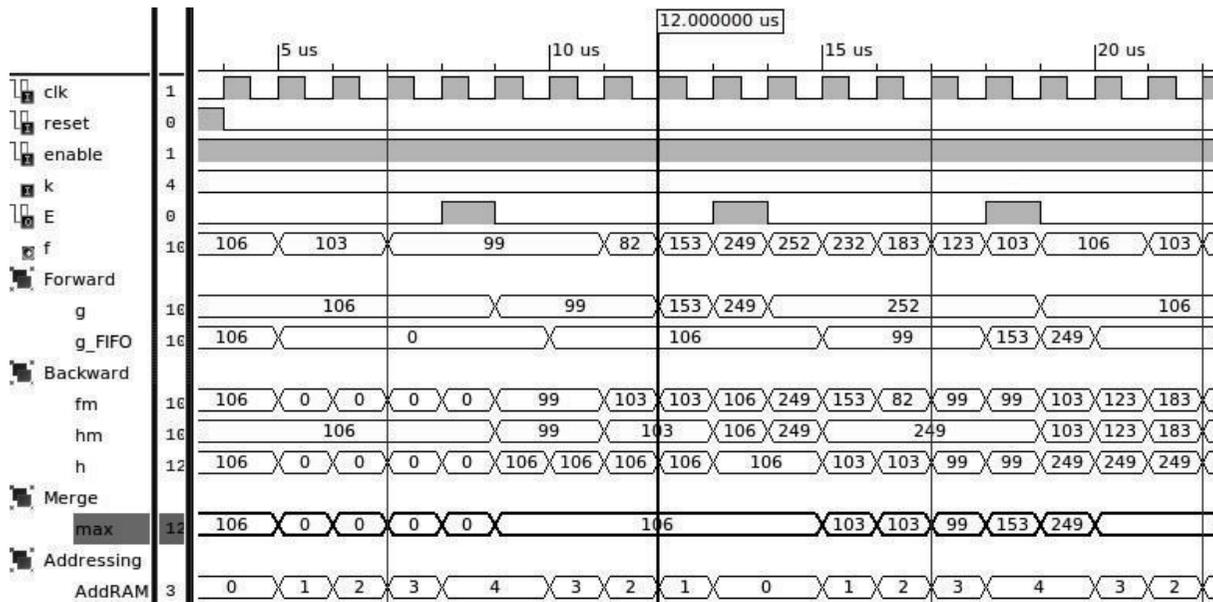


FIGURE 4: Timing diagram snapshot of the architecture functionality for running max filtering over the input pixel stream, f , using a kernel of size $k = 5$. The first output result is produced at 12 microseconds as indicated by the vertical line; then, results are produced on each clock cycle. The signal $AddRAM$ shows the addresses generated through time used for accessing the synchronous single-port memories.

also implemented using a distributed synchronous single-port memory.

Figure 4 shows a time diagram of an 8-bit pixel stream f of an input image used to illustrate the operation of the architecture when a kernel of size $k = 5$ is used. A snapshot of the main signals g and h in the data flow and computation steps for the pipelined architecture are shown in the simulation assuming a clock frequency of 100 MHz. For simplicity, just two control signals E and $AddRAM$ derived from the counter-based controller are shown. Note that $AddRAM$ is generated by reverted address counters and used as addresses to write and read data in the distributed memory. Each stage is active for k consecutive clock cycles and the operation of adjacent stages are delayed for k clock cycles. Signal E indicates the time when a window of the input stream has been processed. As shown in Figure 4, each comparator, after being reset by E , is reused for another adjacent k window.

3.4. Parallelism Enhancement. Because pipelining and parallelism are naturally supported by intrinsic resources of current FPGA devices, it is important to fully utilize these resources to improve performance. At a first level, the proposed architecture was divided into a set of simpler functional elements to carry out the internal computations in a pipelined fashion on the input stream. However, the performance for running max/min filters on two-dimensional signals using rectangular kernels can be improved if function-level parallelism is exploited. Thus, the HGW module can be replicated as much as possible and organized in a more parallel structure as shown in Figure 5, so as to process concurrently several input streams. In this sense, the number of HGW modules depends on the capacity of the target FPGA device and the

actual memory organization that provides data. A set of registers is used to store data coming from the external memory. These registers provide parallel data access to the HGW modules. A multiplexer selects the results produced by the HGW modules and sends them to the output memory.

To apply such parallel scheme for running max/min filters on images, it is assumed that the input image is scanned row by row starting from the upper-left corner sample. In addition, observing that memories can often operate much faster than the user's actual design, memory ports can be time multiplexed to increase the number of independent accesses. In such multipumping scheme [15], the memory system is clocked at a multiple of the main clock, providing the illusion of a multiple port memory. Multipumping brings an area reduction if the external memory speed is significantly higher than the required by the rest of the system. Since the number of required ports or the operating frequency is modest in the proposed design, the main benefit of multipumping is reducing the on-chip memory area at the expense of clock frequency.

4. Implementation and Experimental Results

In this section, experimental results of the FPGA implementation, the hardware resource utilization, and the performance evaluation of the proposed architecture are presented and discussed.

4.1. FPGA Implementation. The Atlys FPGA board from Digilent Inc. has been used for prototyping and VHDL was used as the modeling language. Design parameters such as the kernel size, the image dimensions, and the number of parallel units are parameterizable, so they can be set to the

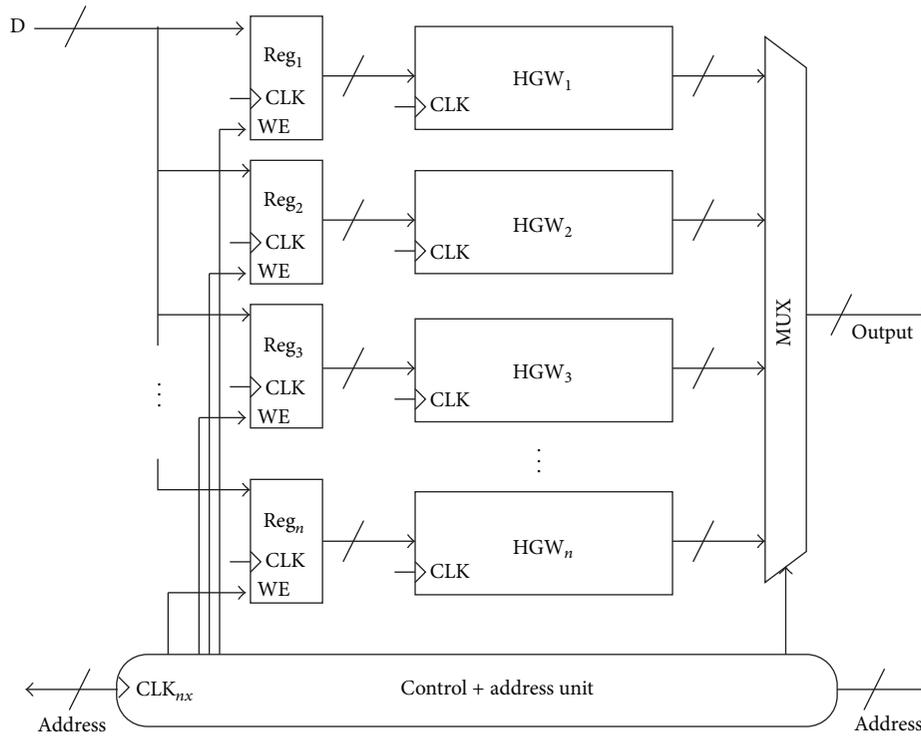


FIGURE 5: Organization of several HGW processing units to exploit function-level parallelism. The address generator unit works at a clock speed n times faster than the computational modules.

appropriate values before synthesis for an optimized implementation. The architecture functionality has been validated on a set of test gray-level images from the Brodatz texture dataset. Figure 6 shows two 1024×1024 test images used in the experiments and the results obtained produced by the architecture using rectangular kernels of different sizes, 21 and 63, for max and min filtering. Larger kernel sizes were also tested and validated but for space consideration are not presented here.

Table 1 shows the FPGA resource utilization and the maximum achievable frequency for three different instances of the architecture using 1, 2, and 4 HGW modules. The presented results are obtained from the reports generated by the Xilinx ISE 13.1 tool suite when the design is targeted to a Spartan-6 device. The entire HGW architecture pipeline fits easily into the device thanks to the use of distributed ram resources. Note that the three logic memory modules used in the design are mapped to LUTs in the FPGA device. Only 96 6-input LUTs are necessary to support any kernel size up to 255, that is, a 256×8 single-port distributed ram. The maximum clock frequency reported by the tool is 250 MHz for a single HGW module with less than one percent of usage of the target device. Thus, potentially a large number of HGW modules can be used without a considerable increase of resource utilization or speed degradation. The hardware resource utilization for a single HGW is similar to the proposed in [13] where 700 slices and 3 Block Rams of 18 Kbits were required. However, recall that authors used an FPGA technology relying on 4-input LUTs and in this work, the used target device natively supports 6-input LUTs; thus, a more

compact implementation is expected. On the other hand, the use of distributed synchronous ram allows to replicate the HGW module so as to increase performance. A post-place-and-route simulation model was used to estimate the power consumption of the proposed architecture using the Xilinx XPower tool. The total power consumption of the 4-HGW design is 0.22 W, dynamic (0.18 W) plus quiescent (0.04 W) power.

4.2. Performance Evaluation. In order to have a baseline for comparison, a straightforward implementation for min/max filtering was carried out in C programming language. Also, the Urbach-Wilkinson algorithm [17] is used for comparison purposes by using the source code provided by authors. Figure 7 shows the computation times for these methods over 2160×1440 gray-scale images. The implementations were carried out on a MacBook Pro with an Intel Core i7 2.66 GHz processor and 4 GB main memory in ANSI C without multithreading and compiled using gcc with O3 optimization flag set. The computation time for the straightforward implementation grows prohibitively large, not being suitable for real-time performance.

Figure 8 shows the near constant processing time, around 25 milliseconds, required for the architecture to filter a 2160×1440 input image for different kernel sizes. Recall that the architecture must operate twice on the input image since it uses kernel decomposition. A single HGW module clocked at 250 MHz processes row by row and then column by column. The architecture takes $3k/2$ clock cycles to produce the first result.

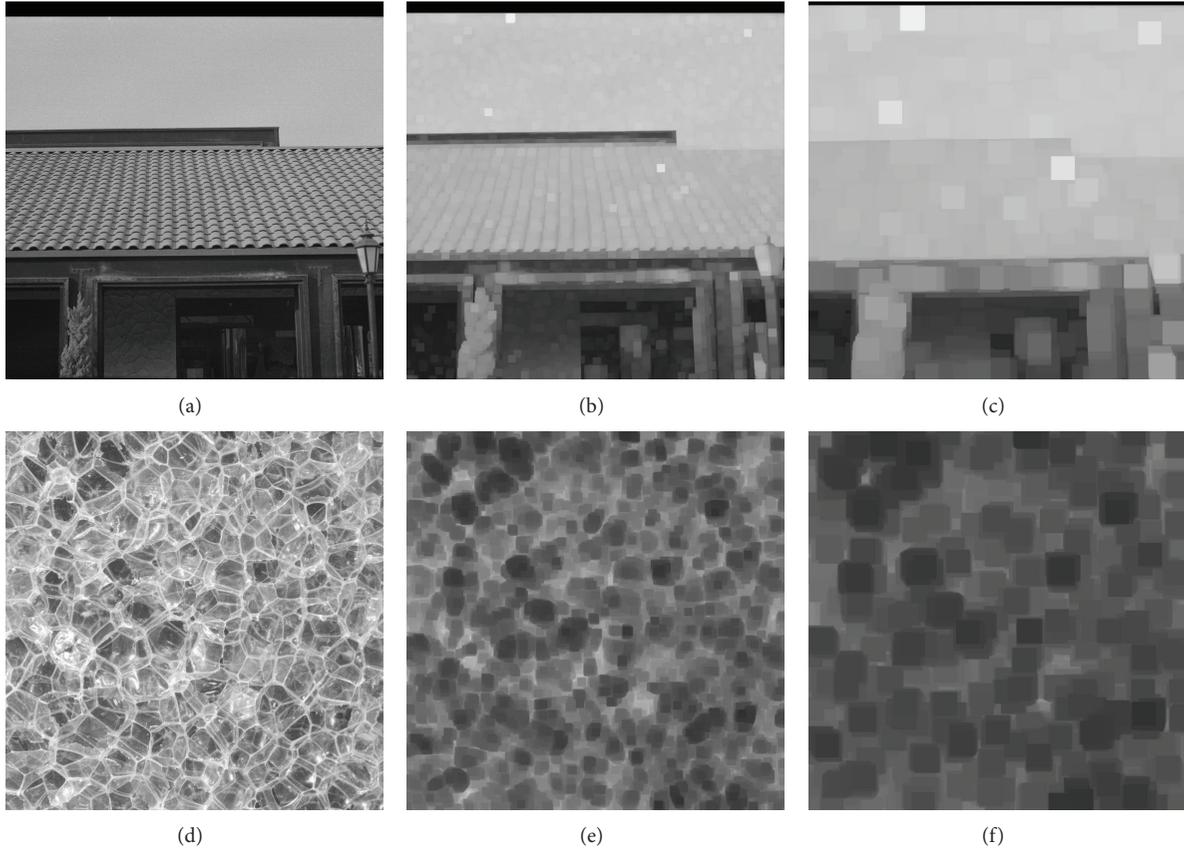


FIGURE 6: Examples of 1024×1024 test images used to validate the architecture functionality. ((a) and (d)) the input images and (b) max filter by 21×21 , (c) max filter by 63×63 , (e) min filter by 21×21 , and (f) min filter by 63×63 .

TABLE 1: Summary of the hardware resource utilization for the proposed architecture targeted to a Xilinx Spartan-6 LX45 device for different number of instances of the HGW module.

Resource utilization (total available)	1-HGW	2-HGW	4-HGW
Slice registers (54576)	75	150	216
Slice LUTs (27288)	258	566	910
LUTs used as logic (27288)	159	368	518
LUTs as memory (6408)	96	192	384
RAMB16BWERS (116)	0	0	0
RAMB8BWERS (232)	0	0	0
Maximum frequency	250 MHz	232 MHz	215 MHz

The processing time required for the MATLAB erosion/dilation optimized implementation is used for comparison purposes. Though it is not a parallel implementation, it forms a useful comparison baseline for this work as it uses the HGW algorithm and implements kernel decomposition. Figure 8 shows that the proposed architecture is faster than the MATLAB optimized implementation, above 10x, with a deterministic response, and also outperforms the Urbach-Wilkinson implementation. The very low resource utilization makes the architecture suitable for embedded applications in low-cost FPGA devices with similar performances as efficient implementations in graphical processing units (GPUs) but with much lower power consumption.

The architecture is able to process images of different sizes and it can be easily extended to improve its performance by the replication of the HGW module. Figure 9 shows the processing times required for max/min filtering for different image sizes and for three degrees of parallelism using 1, 2, and 4 HGW modules clocked at 250 MHz, 225 MHz, and 150 MHz, respectively. Note that when 4 HGW instances are used, it is still possible to achieve more than standard real-time performance, 30 frames per second, even for high-resolution images.

4.3. *Application on Local Adaptive Binarization.* According to the results, the proposed architecture is suitable to

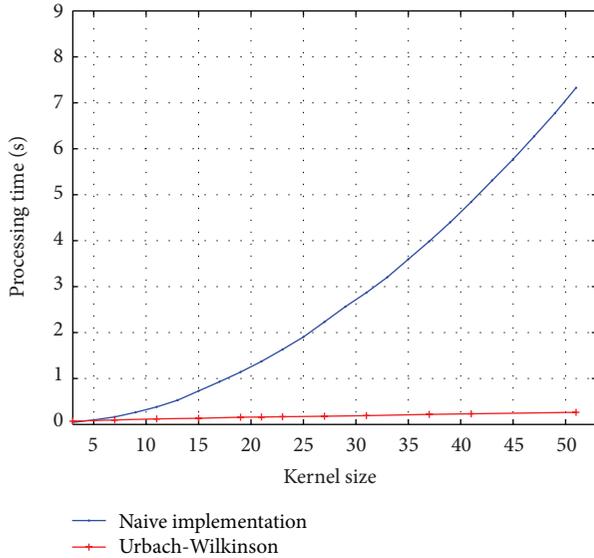


FIGURE 7: Processing time for the running max/min filters on an 2160×1440 input image with different kernel sizes for a straightforward implementation and the Urbach-Wilkinson algorithm.

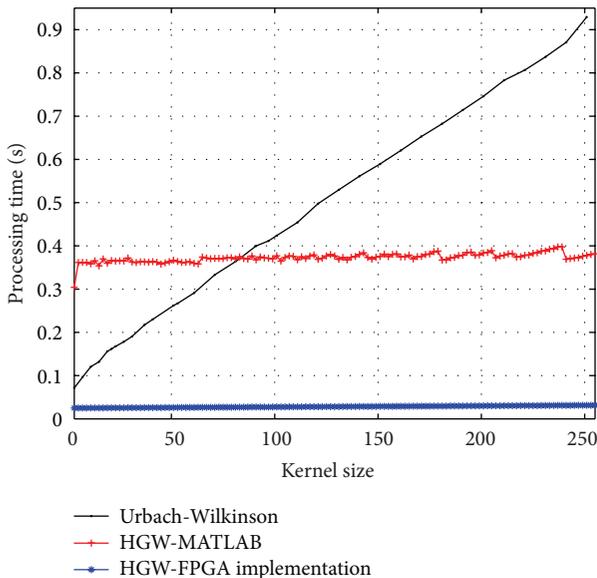


FIGURE 8: Processing time for the running max/min filters on an 2160×1440 input image with different kernel sizes using a single HGW module in the proposed architecture clocked at 250 MHz. The processing time required for the Urbach-Wilkinson algorithm and the MATLAB implementation of the HGW algorithm is also shown for comparison.

be applied in embedded applications thanks to its real-time performance, low resource utilization, and low power consumption. In this section, a further application of the architecture for image binarization is presented. Image binarization converts gray-level or color images into binary ones in order to distinguish objects from background by finding and applying an appropriate threshold for image pixels. In

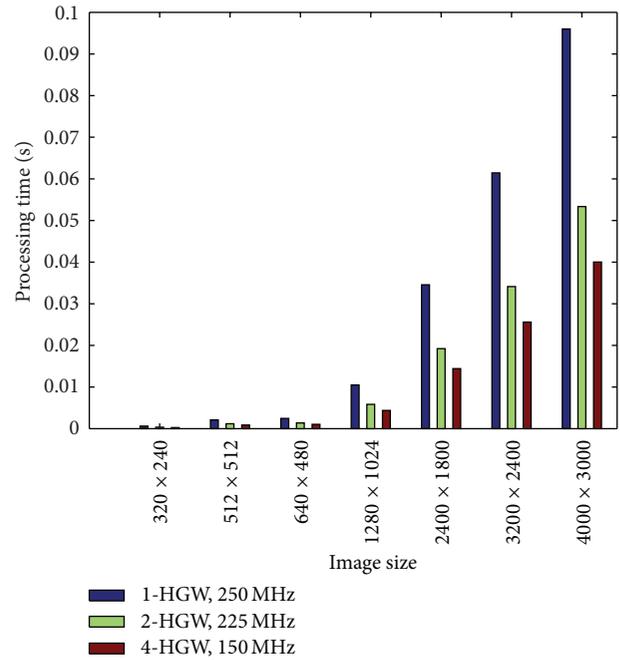


FIGURE 9: Processing time for different image sizes using three different degrees of parallelism.

document image analysis, the main goal is to extract printed characters through optical character recognition (OCR) to analyze relevant textual information from document images from sources such as books, magazines, forms, or newspapers [18]. Local thresholding methods find a threshold for each image pixel based on local characteristics and statistics of pixels within a neighborhood centered around a given pixel [19, 20]. Motivated by the advantages of local adaptive thresholding and by the challenge to handle efficiently the computational cost and the memory bandwidth, proposed architecture has been applied to accelerate computations for the Bernsen algorithm.

In the Bernsen algorithm [21], for each pixel of the original image with gray level $I(x, y) \in [0, 255]$, the local threshold, $T(x, y)$, is set at the midrange value, which is the mean of the minimum and maximum gray level values in a given neighborhood:

$$T(x, y) = \frac{1}{2} [I_{\max}(x, y) + I_{\min}(x, y)]. \quad (5)$$

If the contrast $c(x, y) = I_{\max}(x, y) - I_{\min}(x, y)$ in the neighborhood is below a given threshold k , then it is assumed that the neighborhood consists only of one class, foreground or background, depending on the value of the threshold. Each pixel (x, y) is classified as an object pixel (indicated by value 1) or a background pixel (indicated by value 0) according to the following equation:

$$b(x, y) = \begin{cases} 1, & \text{if } I(x, y) < T(x, y), c(x, y) > k, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

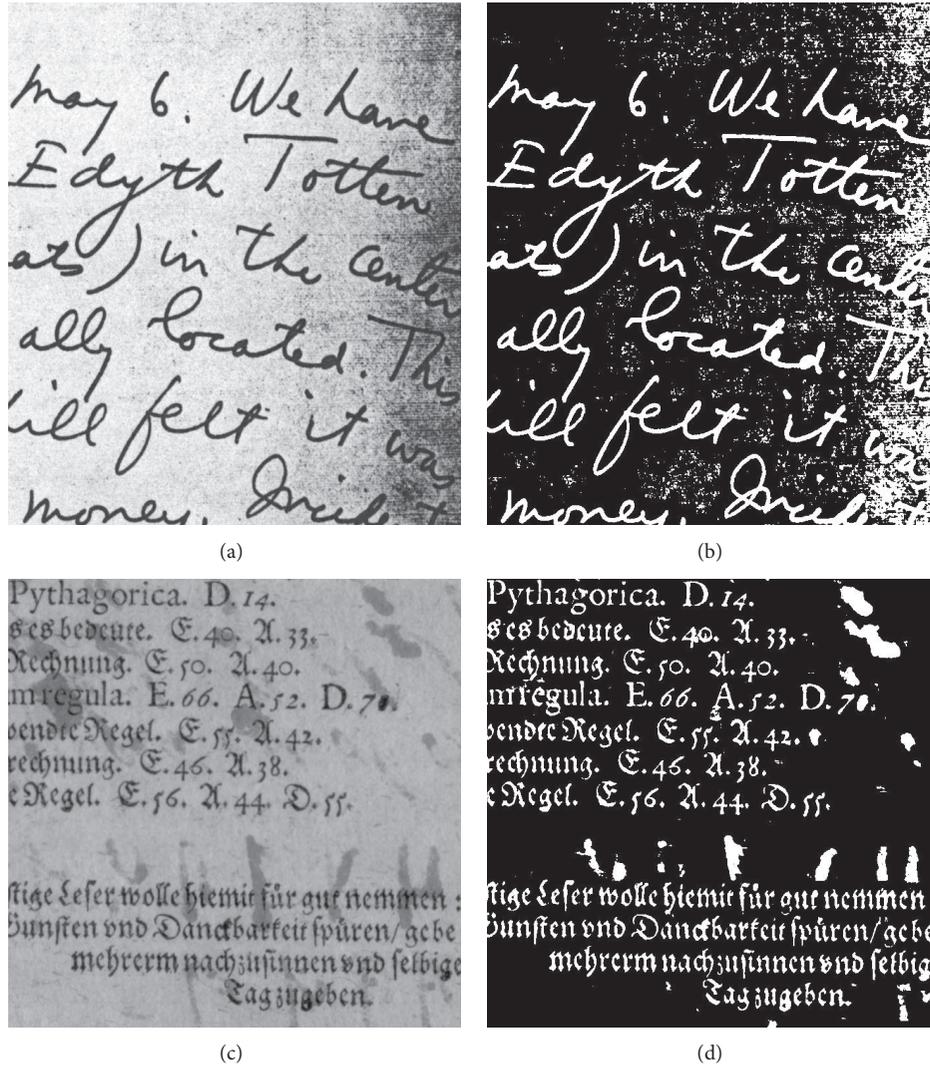


FIGURE 10: Test images used to show the applicability of the architecture for local adaptive thresholding. ((a) and (c)) The input images; ((b) and (d)) thresholded images by using the Bernsen algorithm with a 31×31 window and $k = 60$.

The major part of the computations involved in this method is the calculation of local maximum and local minimum. Thus, the Bernsen algorithm fits well the proposed architecture to speed up computations by using two HGW modules working in parallel. Figure 10 shows two input images and the corresponding binarized ones using a window of 31×31 and $k = 60$. The processing time to binarize a 1024×1024 image with this window size is 8.4 milliseconds, 120 frames per second, at a maximum clock frequency of 250 MHz, when the architecture is targeted to a Spartan-6 device. This yields a throughput over 14 Gpixels/second enough for real-time image processing. Recall, however, that real time is a context relative measure.

5. Conclusions

An efficient implementation of a fast algorithm for arbitrary length max/min filters has been presented. The proposed

architecture is very regular and scalable with a good resource-performance tradeoff suitable to be embedded in low-cost FPGA devices. The proposed design takes advantage of distributed memory resources available in current programmable devices without introducing a high-performance penalty. However, for very large kernel sizes, the area for distributed memory increases rapidly and the operating frequency might drop significantly. This motivates the use of the specialized Block Rams as a more efficient solution. The results show that the proposed implementation could achieve the same throughput with less amount of memory resources compared to the previously reported solution. The architecture, when targeted to a Spartan-6 device, can compute a max/min running filter over a 1024×1024 image with a kernel size up to 255 in 8.4 milliseconds at a maximum clock frequency of 250 MHz. This performance is sufficient for real-time full-HD video processing. The progress of high resolution image applications on embedded systems requires

reviewing existing solutions under this context and proposing hardware accelerators to potentially provide practical, compact, and low power solutions. For future work, it is planned to analyze in detail the power consumption of the proposed implementation, extend further the applicability of the proposed hardware to local adaptive thresholding, and to implement specialized operators in gray-level image morphology.

Acknowledgment

The author kindly acknowledges the partial support received from CONACyT, Mexico, through the Research Grant no. 99912.

References

- [1] M. Brookes, "Algorithms for max and min filters with improved worst-case performance," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 9, pp. 930–935, 2000.
- [2] P. Maragos and R. W. Schafer, "Morphological filters. II. Their relations to median, order-statistic, and stack filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 8, pp. 1170–1184, 1987.
- [3] G. R. Arce and M. P. McLoughlin, "Theoretical analysis of the max/median filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 1, pp. 60–69, 1987.
- [4] P. Maragos and R. W. Schafer, "Morphological filters. I. Their set-theoretic analysis and relations to linear shift-invariant filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 8, pp. 1153–1169, 1987.
- [5] R. Hodgson, D. Bailey, M. Naylor, A. Ng, and S. McNeill, "Properties, implementations and applications of rank filters," *Image and Vision Computing*, vol. 3, no. 1, pp. 3–14, 1985.
- [6] J. Y. Gil and R. Kimmel, "Efficient dilation, erosion, opening, and closing algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1606–1617, 2002.
- [7] H. Yuan and M. J. Atallah, "Running max/min filters using $1+o(1)$ comparisons per sample," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2544–2548, 2011.
- [8] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Pattern Recognition Letters*, vol. 13, no. 7, pp. 517–521, 1992.
- [9] J. Gil and M. Werman, "Computing 2-D min median and max filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5, pp. 504–507, 1993.
- [10] K. I. Diamantaras and S. Y. Kung, "A linear systolic array for real-time morphological image processing," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 17, no. 1, pp. 43–55, 1997.
- [11] C. Torres-Huitzil and M. Arias-Estrada, "Real-time image processing with a compact FPGA-based systolic architecture," *Real-Time Imaging*, vol. 10, no. 3, pp. 177–187, 2004.
- [12] S.-Y. Chien, S.-Y. Ma, and L.-G. Chen, "Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1156–1168, 2005.
- [13] C. Clienti, M. Bilodeau, and S. Beucher, "An efficient hardware architecture without line memories for morphological image processing," *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS '08)*, Springer, Berlin, Germany, vol. 5259, pp. 147–156, 2008.
- [14] O. Déforges, N. Normand, and M. Babel, "Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture," *Journal of Real-Time Image Processing*, pp. 1–10, 2010.
- [15] C. E. LaForest and J. G. Steffan, "Efficient multi-ported memories for FPGAs," in *Proceedings of the 18th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '10)*, pp. 41–50, ACM, New York, NY, USA, February 2010.
- [16] Inc Xilinx, *Using Look-up Tables as Distributed RAM in Spartan-3 Generation FPGAs*, Application Note XAPP464, March 2005.
- [17] E. R. Urbach and M. H. F. Wilkinson, "Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements," *IEEE Transactions on Image Processing*, vol. 17, no. 1, pp. 1–8, 2008.
- [18] Y.-T. Pai, Y.-F. Chang, and S.-J. Ruan, "Adaptive thresholding algorithm: efficient computation technique based on intelligent block detection for degraded document images," *Pattern Recognition*, vol. 43, no. 9, pp. 3177–3187, 2010.
- [19] O. D. Trier and A. K. Jain, "Goal-directed evaluation of binarization methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1191–1201, 1995.
- [20] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146–168, 2004.
- [21] J. Bernsen, "Dynamic thresholding of gray level images," in *Proceedings of IEEE International Conference on Pattern Recognition (ICPR '86)*, pp. 1251–1255, 1986.



Hindawi

Submit your manuscripts at
<http://www.hindawi.com>

