A Configurable Architecture for Fast Moments Computation

Kah-Hyong Chang · Raveendran Paramesran

Abstract In this paper, we present a single-chip architecture for generating a full set of geometric moments using digital filters. Other types of moments such as Zernike and Tchebichef moments can also be implemented. The architecture can be configured for any order of geometric moments and image spatial resolution at run time. The use of a single-scaler method and reusable hardware resources enables higher order moments to be computed. The incorporation of two-level pipelining and masking techniques further increases the throughput. Realized in a field-programmable gate array, the design is capable of processing sixty 512×512 8-bit-pixel images per second at 20 MHz, generating (59+59) orders of geometric moments (3,600 moments). The maximum round-off error is approximately 1 %.

Keywords Image processing · Moments · Digital filters · Real-time · Configurable · High-order · Field-Programmable Gate Array (FPGA)

1 Introduction

Moment invariants first introduced by Hu [1] have been extensively used in object classification, image, shape analyses etc. [2–4]. Teague [3] later introduced a set of continuous orthogonal moments, such as Zernike and Legendre moments. Due to their orthogonality, they found a wider range of applications, which include character recognition, trademark segmentation, and retrieval [5, 6]. Further work has led to the introduction of a new set of discrete orthogonal moments: Hahn, Tchebichef and Krawtchouk moments [7–9]. Fast computational methods have also been developed [10–12]. As a result, applications involving moments of as high as 60 orders are becoming increasingly feasible [13–15]. Notwithstanding the availability of fast algorithms, the computation time still represents a challenge for real-time systems, for their realizations have largely been confined to the software domain.

Since Zernike [3], Tchebichef [8] and Krawtchouk [9] moments are shown to be realizable by computing the Geometric Moments (GM), our motivation has been to realize a run-time configurable, high-order GM computation architecture using the extended digital filter structures. In the next section, we will review the digital filters structures. An overview of the design is provided in Section 3. In Section 4, the single-scaler and cascaded filter structures will be discussed, with the implementation results and performance analysis presented in Section 5. We will conclude the presentation in Section 6.

2 Review of the Digital Filters

GM, m_{pq}, of order $(p+q)$, of a digital image $f(x,y)$ with a spatial resolution of $N\times M$ is defined as:

$$m_{pq} = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)\, x^p y^q. \tag{1}$$

To calculate (1), Hatamian [16] proposed a 2-D filters method to implement the first 16 moments. Wong and Siu [17] moved the delay element in the basic filter structure [16] to the feedback path, for up to the third order. Kotoulas and Andreadis [18, 19] extended the basic filter structure [17], by incorporating a single-pole, feedback type of filter structure as shown in Fig. 1a. The maximum (max) values of *the required*

K.-H. Chang (✉) · R. Paramesran
Department of Electrical Engineering, University of Malaya, 50603 Kuala Lumpur, Malaysia
e-mail: kahhyong@gmail.com

R. Paramesran
e-mail: ravee@um.edu.my

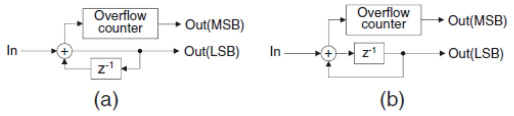


Figure 1 Kotoulas and Andriadis' basic filter structures. **a** Feedback type; **b** Feedforward type.

moment order and/or image spatial resolution (hereinafter referred to as run-time configurable parameters), are limited by the propagation delay along the serially connected adders in the filters. Due to the absence of a registered output for each of the adders, timing critical paths exist for relatively low maximum values of the run-time configurable parameters. When the same design is adapted for column filtering operations, even if all the outputs of the cascaded filter structures are available at the same time, an intermediate memory buffer may still be required for storing their outputs prior to the matrix multiplication operations. As memories have limited data ports, delays are inevitable anyway.

Later, Kotoulas and Andreadis [20] improved on the filter structure in [16]. A single-pole, feedforward type of the filter structure shown in Fig. 1b is used. The propagation delay can now be ignored, as the outputs of the adders are registered. However, the size of the accumulator grid structure still increases significantly with the increasing maximum order of moments [18–20], and presents challenge to the routing. Other drawbacks include the insufficient time for the filters to output the results before the next row of input data arrives. Next, we introduce a design to address the aforementioned shortcomings.

3 Overview of the Design

The top-level diagram in Fig. 2 comprises the two main modules of the design: the digital filter and the matrix multiplication modules. The modules are connected by an intermediate Random-Access Memory (RAM), which stores the intermediate data and scale-factors. In this design, the row adders and other associated circuitry are collectively known as the cascaded filter structures. As is in [16], images need to be first reversed.

An external host ensures that valid run-time configurable parameters have been written before asserting the start row filtering signal. The moment generator issues the input ready signal to the host to signal its readiness for the next image. The host indicates the arrival of the next image by asserting the start row filtering signal. By using this handshaking, and storing the intermediate data, we have discarded the column adders and time-shared the row adders. At the end of the column filtering operation, the cascaded filter structures output the final filter output data to the intermediate RAM, that is accessed by the matrix multiplication module, which incorporates a multiplier, an accumulator, and a coefficient generator that generates the coefficients [21] used in the two phases of matrix multiplication. After a given latency that depends on the run-time configurable parameters, the moment values can

Figure 2 Top-level architecture.

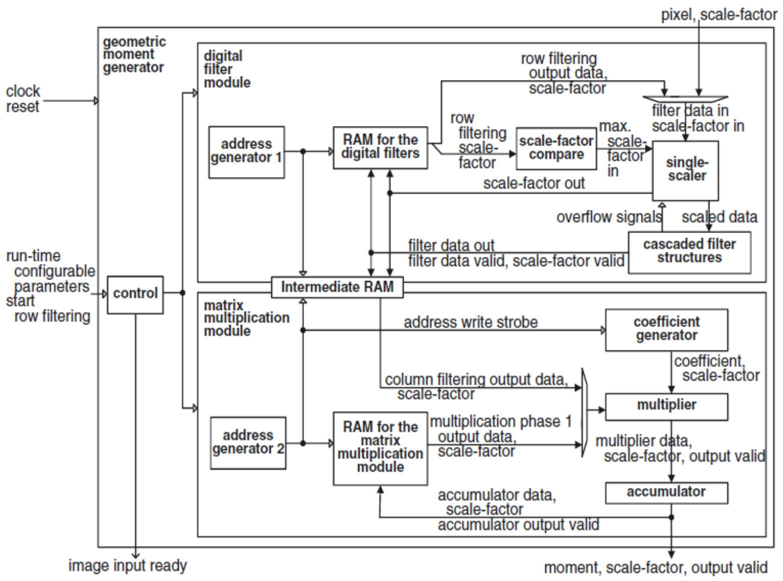
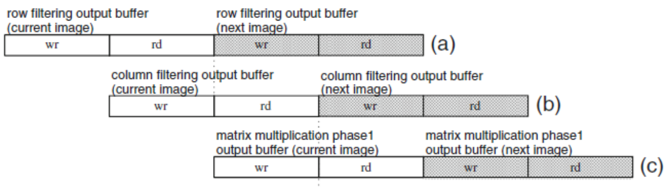

pixel, scale-factor
geometric moment generator
digital filter module
clock
reset
row filtering output data, scale-factor
filter data in
scale-factor in
address generator 1
RAM for the digital filters
row filtering scale-factor
scale-factor compare
max. scale-factor in
single-scaler
scale-factor out
overflow signals
scaled data
run-time configurable parameters
start row filtering
filter data out
filter data valid, scale-factor valid
cascaded filter structures
control
Intermediate RAM
matrix multiplication module
address write strobe
coefficient generator
coefficient, scale-factor
column filtering output data, scale-factor
multiplier
address generator 2
RAM for the matrix multiplication module
multiplication phase 1 output data, scale-factor
multiplier data, scale-factor, output valid
accumulator
accumulator data, scale-factor
accumulator output valid
image input ready
moment, scale-factor, output valid

Figure 3 Buffer occupancies of: **a** the RAM for the digital filter module; **b** the intermediate RAM; **c** the RAM for the matrix multiplication module.


row filtering output buffer (current image)
row filtering output buffer (next image)
wr
rd
wr
rd
(a)
column filtering output buffer (current image)
column filtering output buffer (next image)
wr
rd
wr
rd
(b)
matrix multiplication phase1 output buffer (current image)
matrix multiplication phase1 output buffer (next image)
wr
rd
wr
rd
(c)

be read at the moment and scale-factor output ports. The moment values are accompanied by the output valid signals. The design features that are used to accomplish the aforementioned operations are described next.

3.1 Data Buffering and Time-Sharing of Resources

In order to achieve flexibility in configurability and enable main hardware resources to be time-shared, the intermediate and output data from the filters are stored in the RAM buffers. Figure 3a, b and c show the buffer occupancy for the RAM for the digital filters (row filtering output buffer), the intermediate RAM (column filtering output buffer), and the RAM for the matrix multiplication module (matrix multiplication phase 1 output buffer) respectively. By storing the filter intermediate outputs in the RAM for the digital filters, the cascaded filter structures are time-shared between the row and column filtering. External memory resources may also be used, by providing suitable interfaces and controllers. The removal of the column filters results in configurability, and data-scaling can be carried out by using only one scaler.

3.2 Data Scaling

By scaling the intermediate data, the size of the datapath circuit elements such as adders is no longer rigidly defined. Figure 4 shows a single-scaler with the cascaded filter structures. The scaling begins at the input before the first filter structure in the cascaded filter structures. The use of a single-scaler is feasible for designs based on the feedforward type of basic filter structure [16], but not for the feedback type [17]. The following illustrations will explain the reasons why this is the case.

In Fig. 5, m represents the maximum order of moment, whereas n represents time instance. The input to the filter is x, which, in this illustration is equal to 1 for four consecutive clock cycles, starting from $n=1$, and the outputs of each of the $(m+1)$ cascaded filter structures are y_0, y_1,.., y_{m-1}, y_m, respectively. Figure 5a is an illustration for the feedback type of cascaded filter structures. The outputs (bold-font values) of the filter structures are shown in Fig. 5a, at $n=4$, for $m=3$, given by:

$$y_r(n) = x(n) + \sum_{k=0}^{r} y_k(n-1), \tag{2}$$

where y_r is valid at $n=m+1$, $r=\{0,1,..,m-1,m\}$.

During a particular time instance n, any of the addition operations in (2) may result in an overflow. Maximum $(m+1)$ overflows and $(m+1)$ scale-factors are required to keep track of the output values of the filtering stages, $y_r(n)$. In addition, $(m+1)$ scale-factors *compare* operations are required to compare and determine which of the corresponding output values of the filtering stages needs to be scaled. Maximum $(m+1)$ *right-shift* operations must take place within one cycle. It is therefore insufficient to use only a single-scaler.

The feedforward type of cascaded filter structures and the state table is depicted in Fig. 5b, in which the outputs of the filters are in as bold fonts, at $n=\{5,6,7,8\}$, for $m=3$, given by:

$$y_r(n) = x(n-(r+1)) + \sum_{k=0}^{r} y_{r-k}(n-(k+1)), \tag{3}$$

where $y_r(n)$ is valid at $n=m+r+2$, $r=\{0,1,..,m-1,m\}$.

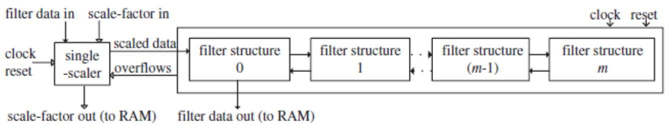
If we rearrange (3), we obtain:

$$y_r(n) = B + y_r(n-1), \quad B = \begin{cases} x(n-1) & r=0 \\ y_{r-1}(n-1) & r=\{1,2,..,m-1,m\}, \end{cases} \tag{4}$$

where $y_r(n)$ is valid at $n=m+r+2$.

In (2), the inputs of any two-input *add* operations in the summation operation depend on the outputs of other *add* operations within the same operation at the same time instance. However, (4) shows that the inputs of a two-input *add* operation do not depend on the outputs of other *add* operations at the same time instance. If any of the *add* operations overflows, the right-shift-by-one-bit-and-round of the

Figure 4 Cascaded filter structures and the single-scaler.



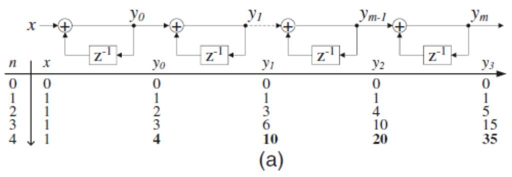
output of that particular *add* operation will not affect the other *add* operations taking place at the same time instance; one scale-factor is already sufficient to represent the radix-2 exponent of all the outputs of the filtering stages at that particular time instance. With $y_r(n)$ right-shifted by only one bit, neither a comparator nor right-shifter (for right-shifting an operand for more than one bit) is necessary.

3.3 Two-Level Pipelines

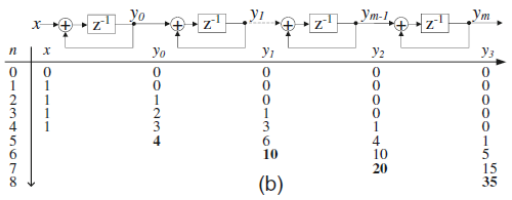
Two levels of pipelines are used in order to increase the throughput. The architecture does not suffer from the drawback as in [16], where the filter does not have enough time to generate results before the next row of input data arrives. To overcome this, we incorporate a micro-level pipeline in the filter structure. First, we introduce an extra output data register in each filter structure. When these registers are connected together as the filter structures are cascaded, they form a chain of one-in-serial-out shift registers. The difference compared to the normal kind of parallel-in-serial-out shift registers is that, instead of updating the shift registers in parallel in one clock cycle, only one of the shift registers is updated with an output from the filtering structures in a clock cycle during the filtering stage.

By using the one-in-serial-out shift registers and the data buffering method, and by time-sharing the row adders for both row and column filtering operations, the output data obtained from a previous filtering operation can be shifted out and stored into the RAM serially, while the current filtering operation is still active, without the need of using a large selector to select the data. However, there is still a gap of inactivity in the filter pipeline timing diagram, labeled as *gap* in Fig. 6. After introducing another output data register in each of the filter structures, forming a second chain of one-in-serial-out shift registers, the gap is closed. The filtering operations are now fully pipelined. In terms of chip area savings, with the one-in-serial-out shift registers incorporated into the design, the use of large-size multiplexers becomes unnecessary. The one-in-serial-out shift registers also contribute to the ease of configuring the required order and parameterizing the maximum order.

On another note, the flow of data through the digital filter and matrix multiplication modules as shown in Fig. 2 is buffered by the intermediate memory buffers. Figure 3a and c show that this macro-level pipelining technique has enabled the filtering and the multiplication operations to be executed concurrently. The resulting increase in throughput indirectly translates into higher run-time configurable parameters. Nevertheless, if the ratio of image spatial resolution to the maximum order of moments is close to one, the matrix multiplication operation may take longer time to complete than the digital filtering operation. To coordinate the macro-level pipeline, inter-modular handshaking signals are used as semaphores.



n	x	y_0	y_1	y_2	y_3
0	0	0	0	0	0
1	1	1	1	1	1
2	1	2	3	4	5
3	1	3	6	10	15
4	1	**4**	**10**	**20**	**35**



n	x	y_0	y_1	y_2	y_3
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	2	1	0	0
4	1	3	3	1	0
5		**4**	6	4	1
6			**10**	10	5
7				**20**	15
8					**35**

Figure 5 Cascaded basic filter structures and the state tables illustrated for $m = 3$. **a** Feedback type; **b** Feedforward type.

3.4 Coefficient Generator

The use of the coefficient generator as shown in Fig. 2 facilitates the parameterization of the maximum order of the GM, as the coefficient generator auto-generates the coefficients [21] used in the matrix multiplications on-the-fly. Updating the Read-Only Memory (ROM) table when a higher maximum order of moment is implemented is no longer necessary. Two dual-port RAMs are used in the coefficient generator. Their size increases linearly with the maximum order. While one RAM is used in providing the coefficients for matrix multiplications as well as for generating the new set of coefficients, the other is used for storing the derived set of new coefficients. No multipliers are used; iterative additions are performed instead. It is noted that the coefficient generator can be replaced by the ROM table, without affecting the configurability. However, the increase in size of the ROM table is proportional to the square of the amount of increase in the maximum order of moments. Therefore, as far as parameterization and chip size considerations for higher order of moments are concerned, the coefficient generator is an attractive option. The operations and structures of the design features will be discussed next.

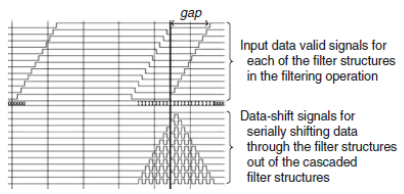


Figure 6 Timing diagram showing the gap between consecutive filtering operations.

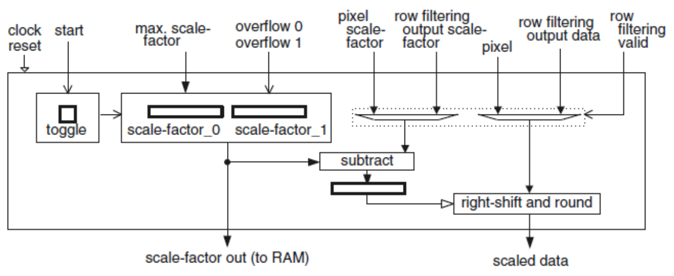


Figure 7 Single-scaler design.

4 Operations of the Single-Scaler and Filter Structures

A maximum scale-factor is obtained from the scale-factor compare module in Fig. 2. The single-scaler receives this value. One scaler is sufficient, even though there can be a maximum of two filtering operations in the cascaded filter structures pipeline.

At the start of the filtering operation, either one of the two scale-factor registers in Fig. 7 is used to store the maximum value. When a new filtering operation begins, the toggle flag is inverted. This toggle flag is implemented because two filtering operations may co-exist at the transitions between two consecutive row or column operations. Depending on the toggle value, one of the two scale-factors selected by a multiplexer is incremented by 1 if an overflow situation occurs, while the other is incremented by 1 if an overflow situation occurs in a previous filtering operation that may still be present in the pipeline. The updated former scale-factor is subtracted by the scale-factor associated with an incoming data. The results will determine the number of bits the data are to be right-adjusted, after which they are output to the cascaded filter structures.

In the cascaded filter structures shown in Fig. 8, there is a total of $(m+1)$ cascaded filter structures. Each of the filter structures is identical. The first filter structure in the cascaded filter structures receives the scaled data from the single-scaler, and a filter input valid signal from the control module. The cascaded filter structures output two signals: overflow 0 and overflow 1 to the single-scaler shown in Fig. 7, as well as back to each of the filter structures. These signals indicate to the single-scaler the overflow situations of the two filtering operations. Two filtering operations

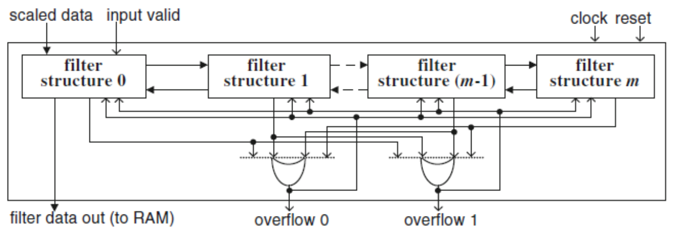


Figure 8 Cascaded filter structures design.

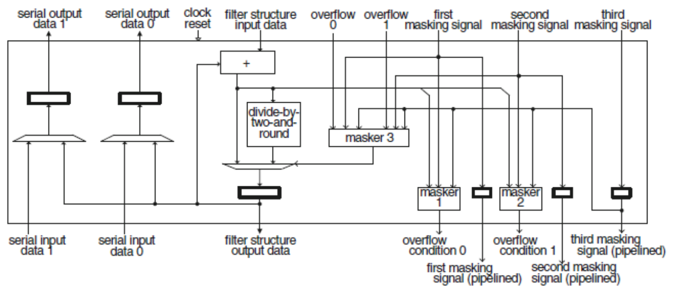


Figure 9 Filter structure and the overflow signals.

Table 1 Top-level parameters.

Image spatial resolution	512×512	Matrix multiplication (MatMul) input data	270b
Scale-factors	14b	MatMul coefficients	33b
Digital filter operands	270b	MatMul accumulator input data	302b
Input pixels	8b	Max orders of moment	59+59

may be present in the micro-pipeline of the cascaded filter structures at the same time.

The design of the filter structure is as depicted in Fig. 9. It has a divide-by-two-and-round circuit to cater to an overflow situation, which is deemed to have occurred if any of the filter structures pertaining to the same filtering operation in the cascaded filter structures has an overflow condition. Since a filter structure in a consecutively connected part of the cascaded filters may belong to either one of the two micro-pipelined filtering operations, its overflow signal is masked by the two signals pipelined down the filter structures. The first masking signal indicates that the current operation in that filter structure is pertinent to the first filtering operation in the micro-pipeline, as the second masking signal is to the second filtering operation.

Each filter structure in the consecutively connected part of the cascaded filters is activated by a third masking signal that has been pipelined down before the moment computations start, for a number of clock cycles depending on the order of moment configured by the host. This masking signal masks out the overflow signals if that filter structure is not activated.

Referring to Fig. 9, the left-most section of the filter structure contains two data registers. When cascaded together, they form two one-in-serial-out chains. These chained registers store the outputs of the filter structures shown as bold-font values in Fig. 5b, as discussed in Section 3.2.

5 Implementations and Performance

The circuits are modeled in Verilog HDL. In order to find an optimal set of top-level circuit parameters, the design is parameterized using increasingly demanding settings. Table 1 lists the final parameters selected.

Table 2 Maximum round-off errors in the GM.

Stream	Max error (%)
Baboon	1.27
Barbara	1.08
Bridge	1.49
Boat	0.56

Table 3 FPGA (EP4SE530F43C2ES) implementation results.

Combinational ALUTs	110,580/424,960
Dedicated logic registers	51,388/424,960
Total block memory bits	11,564,928/21,233,664
Fmax1 , *Fmax2*	29.7 MHz, 20.38 MHz

The maximum round-off errors in the GM obtained by comparing the functional simulation results with the reference results generated by the GNU C++ (with GNU Multiple Precision Arithmetic Library), are shown in Table 2. When *0xFF* is fed in as inputs, the maximum round-off error is 0.91 %.

The implementation results generated by *Altera* Quartus II 9.1 are summarized in Table 3. The logic utilization is 39 %. Two clock domains are specified for the synthesis, for the digital filters as well as matrix multiplication modules. Their maximum frequencies are *Fmax1* and *Fmax2* respectively, which show that the critical path is located in the matrix multiplication module.

Using only one operating frequency of 20 MHz, the latencies of the main operations involved are listed in Table 4, which shows that the performance of the digital filters is the bottleneck.

Sixty images of 512×512 grayscale pixels are processed by the design in 0.972 s. In total, (59+59) orders of GM are generated in 16.2 ms per image. Extending the feedforward type of accumulator grid structures [18–20] to obtain a full set of 3,600 moments, a fair comparison can then be made in terms of the latencies of the digital filtering operation. Since the matrix multiplication operation in [18–20] is performed off-chip, Tables 5 and 6 show only the figures for the digital

Table 4 Latencies of the operations in moments computation.

Operation		Latency (cycles)	
Filtering	Row, Column	262,144+61,746	323,890
MatMul	Phase 1, 2	110,045+110,035	220,080

Table 5 Latencies of the digital filtering (filt.) operation and run-time configurability of the architectures.

	Row filt.	Col filt.	Total (cycles)	Configurability
[18–20], if bit-serial adders used in column (col) filters	$N(M+q+1)$	$M(p+1)$	323,584	No
[18–20], if ripple-carry adders used in col filters		$(p+1)$	292,924	No
[22]	$NM+(q+1)$	N.A.	262,204	No
This work	$NM+(q+1)$	$(q+1)(N+p+1)$	296,524	Yes, in powers of 2

Link to Full-Text Articles :

http://link.springer.com/article/10.1007/s11265-013-0857-9