26

FPGA Based Acceleration for Image Processing Applications

Griselda Saldaña-González and Miguel Arias-Estrada
Computer Science Department
National Institute for Astrophysics, Optics and Electronics (INAOE)
Puebla, Mexico

1. Introduction

Image processing is considered to be one of the most rapidly evolving areas of information technology, with growing applications in all fields of knowledge. It constitutes a core area of research within the computer science and engineering disciplines given the interest of potential applications ranging from image enhancing, to automatic image understanding, robotics and computer vision. The performance requirements of image processing applications have continuously increased the demands on computing power, especially when there are real time constraints. Image processing applications may consist of several low level algorithms applied in a processing chain to a stream of input images. In order to accelerate image processing, there are different alternatives ranging from parallel computers to specialized ASIC architectures. The computing paradigm using reconfigurable architectures based on Field Programmable Gate Arrays (FPGAs) promises an intermediate trade-off between flexibility and performance (Benkrid et al., 2001).

The present chapter is focused on how a well defined architecture can deliver high performance computing in a single chip, for image processing algorithms, in particular those based on window processing, i.e. convolution. The core architecture is a parallel processors array that can be the basis for processing several image algorithms based on window processing. The architecture is targeted to a single medium size FPGA device following the reconfigurable computing paradigm. The idea is to propose a platform that allows the acceleration of the computationally demanding part of a family of image processing algorithms.

The architecture introduces a new schema based on the use of local storage buffers to reduce the number of access to data memories and router elements to handle data movement among different structures inside the same architecture. These two components interact to provide the capability of processes chaining and to add flexibility to generalize the architecture functionality in order to constitute a versatile and scalable hardware platform.

The architecture copes with window-based image processing algorithms due to the fact that higher level algorithms use the low-level results as primitives to pursue cognitive level goals.

The contribution shows several variations of the architecture for convolution, 2D-filtering and motion computation. The motion computation correlation based algorithm and

Source: Image Processing, Book edited by: Yung-Sheng Chen,
ISBN 978-953-307-026-1, pp. 572, December 2009, INTECH, Croatia, downloaded from SCIYO.COM

architecture are further detailed in order to show the flexibility on one of the most computational demanding algorithms in image processing.
The obtained results show the benefits that can be provided by a system implemented with FPGA technology and reconfigurable computing, since a high degree of parallelism and a considerable hardware resource reutilization is reached. Furthermore, with a standard medium size FPGA, a peak performance of 9 GOPS can be achieved, which implies operation in video rate speed.
Finally in this chapter some conclusions are presented emphasizing the key aspects of this approach to exploit both spatial and temporal parallelism inherent in image processing applications. The contribution concludes with some guidelines learned from the architecture design exploration. New opportunities, recommendations and future work are discussed.

2. Low-level image operators

Low-level image processing operators can be classified as point operators, window operators and global operators, with respect to the way the output pixels are computed from the input pixels (Umbaugh, 1998).
A window-based image operator is performed when a window with an area of w×w pixels is extracted from the input image and it is transformed according to a window mask or kernel, and a mathematical function produces an output result (Li & Kunieda, 1999). The window mask is the same size as the image window and their values are constant through the entire image processing. The values used in the window mask depend on the specific type of features to be detected or recognized. Usually a single output data is produced by each window operation and it is stored in the corresponding central position of the window as shown in Fig. 1.

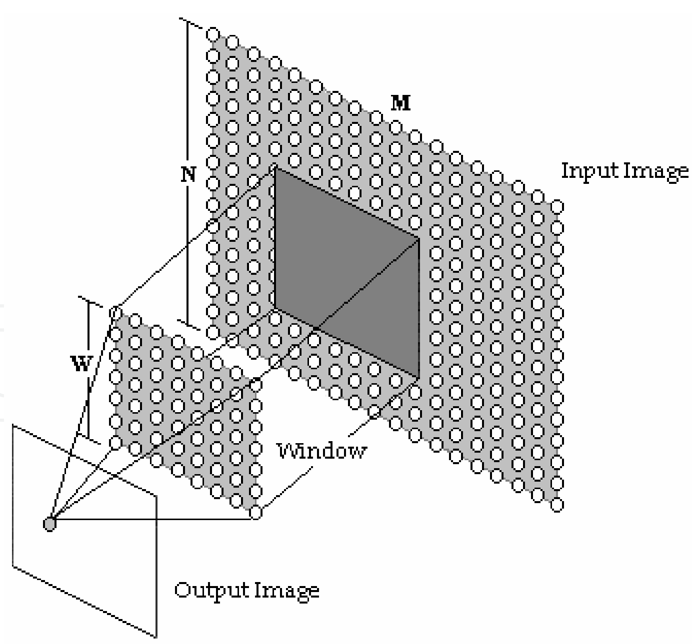


Fig. 1. Schematic representation of a window based operation

Window-based operations can be formalized mathematically as follows. Let I be an $M×N$ input image, Y the output image, and W a $w×w$ window mask. A window operation can be defined by Equation (1):

$$Y_{rc} = F(f(W_{ij}, I_{r+i,\ c+j})) \quad \forall (i,j) \in w \times w,\ \forall (r,c) \in M \times N \tag{1}$$

Where w_{ij} represents a coefficient from the window mask W, $I_{r+i,\ c+j}$ represents a pixel from a $w \times w$ window around the (r, c) pixel in the input image, f defines a scalar function, and F defines the local reduction function.
Window-based operators are characterized because the same scalar function is applied on a pixel by pixel way to each individual pixel of one or more input images to produce a partial result. Common scalar functions include relational operations, arithmetic operations, and logical operations. The local reduction function reduces the window of intermediate results, computed by the scalar function, to a single output result. Some common local reduction functions employed are accumulation, maximum, and absolute value. Scalar and local reduction functions form the image algebra to construct window-based image applications. In order to implement a flexible architecture these functions are considered (Torres-Huitzil & Arias-Estrada, 2005); (Ballard & Brown, 1982); (Bouridane et al., 1999).

3. Architecture description

The rectangular structure of an image intuitively suggests that image processing algorithms map efficiently to a 2D processors array, therefore the proposed architecture consists of a main module based on 2D, customizable systolic array of $w \times w$ Processing Elements (PEs) as can be observed in Fig. 2 diagram.
The main purpose of the architecture is to allow processes chaining, therefore the basic scheme shown in Fig. 2, can be replicated inside the same FPGA several times in order to process different algorithms independently. This processes chaining scheme provides the advantage of using a reduced bandwidth for communication between processing blocks since all of them are inside the same FPGA.

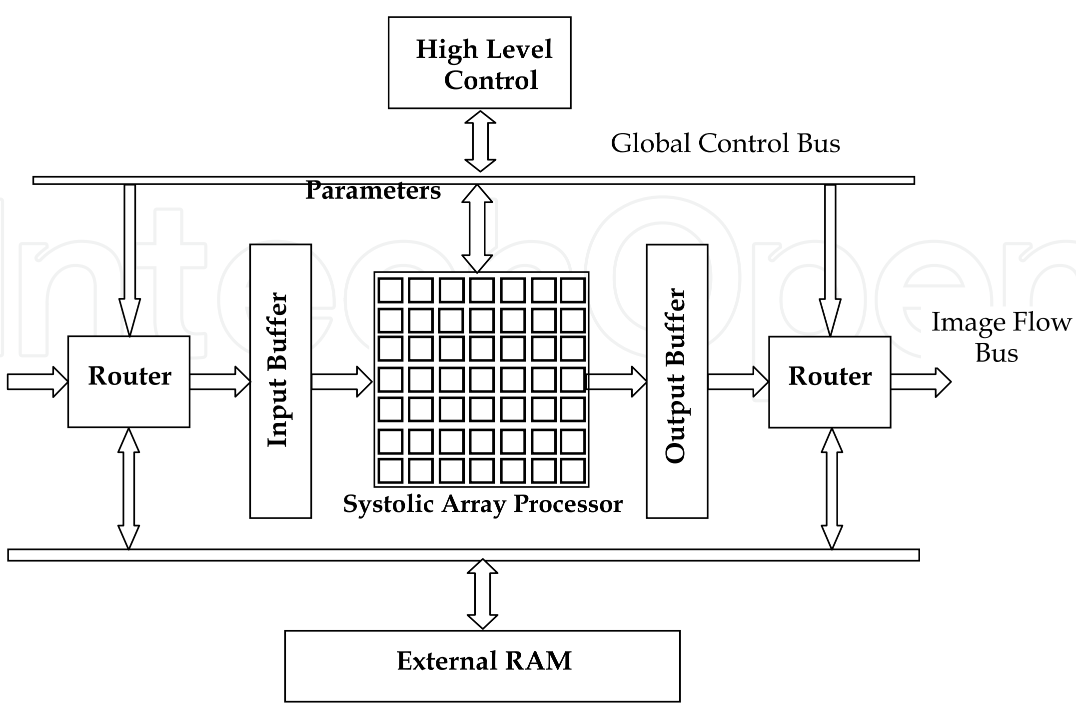


Fig. 2. Block diagram of the architecture

The simplified block diagram of the architecture shown in Fig. 2 comprises six main blocks:

- A high level control unit
- An external main memory
- A dedicated processor array
- Routers
- Image buffers
- Internal buses

High Level Control Unit: This unit could be placed in a host PC or embedded in the FPGA. The main purpose of the control unit is to manage the data flow and synchronize the different operations performed in the architecture. The high level controller starts and stops the operation in the system, furthermore, it is responsible of image capturing and displaying. From the PC it is possible to choose a particular operation that can be performed by the PEs in the systolic array, to coordinate operations and to manage bidirectional data flows between the architecture and the PC. From this unit, the user can select configuration parameters to customize the architecture functionality; the parameters include the size of the images to be processed, the coefficients for the mask to be used during processing and the kind of arithmetic to be employed between integers or fixed-point.

Main Memory: The memory in the architecture is a standard RAM memory for storing data involved in the computations. The data in the memory are accessed by supplying a memory address. The use of these addresses limits the bandwidth to access the data in the memory, and constrains the data to be accessed through only one memory port. Furthermore, the time to access the data is relatively long, therefore a buffer memory is included to store the data accessed from memory and to feed the processor array at a much higher rate. The buffers are used to re-circulate the data back to the processors, and they reduce the demand on main memory. An important issue to be solved is the allocation of area to implement data buffers. To obtain good performance one of the issues in the architecture design is, therefore, how to schedule the computations such that the total amount of data accesses to main memory is bounded.

Processor Array: The processor array is the core of the architecture where the PEs are organized in a 2-D systolic approach; and where the algorithms are executed. The processor array obtains image pixels from the buffers, and mask coefficients from memory to start a computation cycle. The processing array achieves a high performance due to a pipelined processing schema and local connections without long signal delays. The array organization with a small number of boundary (I/O) processors reduces the bandwidth between the array and the external memory units. The control unit specifies and synchronizes the actions to be performed in the PEs.

Routers: The Router unit is responsible for all data transfers in and out of the systolic array as well of interfacing processing modules to external memories. The data streams routers take data from/to input/output image memories and make explicit the data parallelism usually found in the image processing. The incoming data is stored in external memory RAM and data is brought into a set of internal buffers prior to be processed in parallel. The processed data by a processing block can be stored and then transmitted to an external memory output using a router.

Buffers: The purpose of the buffers is to supply data to the processors array and mask the long main memory latencies. The buffers have a fixed amount of storage to keep some rows of the input image or the intermediate data from a processing module. The storage buffers are organized in a First-Input, First-Output (FIFO) style. In each clock cycle, the data present at the buffers are sent to the processors array or to the main memory. Address decoding for the buffer is carried out using pointers that make reference to the buffer row that is being processed or being filled. These pointers allow a circular pattern in data movement inside the buffers. The buffer basically performs the following operations:

- Pre-fetches data from the main memory into its rows to hide the memory latency
- Reorders the information according to the processing needs of the algorithm to increase parallelism
- Stores intermediate information for its reutilization in subsequent processing blocks

Internal Buses: The global bus interconnects architecture elements to interchange back and forward control or configuration information, i.e. mask coefficients. In addition, this bus is connected to the high level control unit placed in a Host processor which is in charge of data and parameters transfer via Direct Memory Access (DMA) with the processor.

This architecture schema resembles a high level pipeline representation, formed of memory units and computing units. The architecture is intended for data communication among processes using data buffers abstraction. With these elements it is possible to chain processes since different processing blocks inside the same FPGA can carry out a different window-operator over the same data set. The results obtained by each block can be stored in the output image buffers and reused by subsequent processing blocks. This structure of cascading interconnection is a key feature of the architecture since it supplies generality to the array of processors, providing enough flexibility to run a variety of low-level processing algorithms and constitutes a platform to pursue the implementation of higher complexity algorithms.

3.1 Systolic array

The processor block of the architecture is shown in Fig. 3. In our implementation, the systolic array is a 7×7 set of configurable PEs. A window mask corresponds to the whole array, with every PE representing a pixel from the input image. The PEs array is vertically pipelined, PEs are activated progressively every clock cycle as shown in Fig. 4.

At every clock cycle all PEs in an array column receive the same column of image pixels but mask coefficients are shifted from left to right between the array columns to calculate the window operation. Partial results are shifted to a Local Data Collector (LDC) in charge of accumulate results located in the same column of the array and the captured results are sent to the Global Data Collector (GDC). The GDC stores the result of a window processed and sends it to the output memory buffer.

After a short latency period, all PEs in the array are performing a computation according to a control word. From that moment on, each new column of pixels sent to the array shifts the window mask to a new adjacent position until the whole image has been visited in the horizontal direction.

If reading image pixels from the buffer one row below, it is possible to cross the image in the vertical direction. The image buffer is updated during PEs operation, in a circular pipeline schema.

This image buffer was implemented with two port BlockRAM memories, where image pixels are stored as neighboring elements.

Routers take data from the input image memories and transfer them to the input buffers that store as many rows as the number of rows in the mask used for processing a window. An additional row is added to the buffer to be filled with new image data in parallel with the rows being processed; in this way the memory access time is hidden. Each time a window is

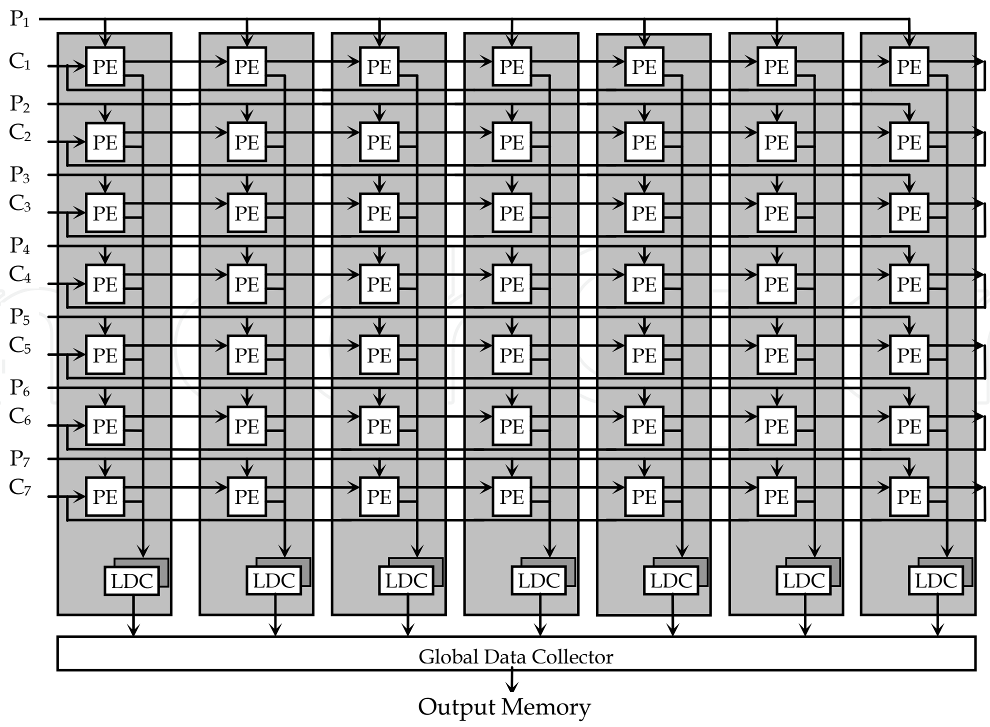


Fig. 3. 2D systolic array implementation

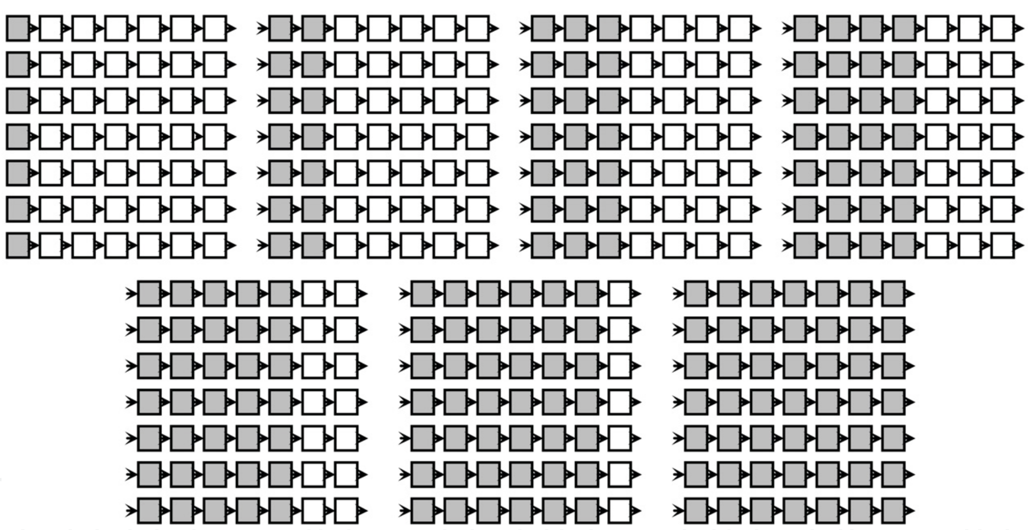


Fig. 4. PEs activation schema

slid in the vertical direction, a new row in the buffer is chosen to be refreshed with input image data, following a FIFO style. When the buffer end is reached, the first buffer row is reused following in this way the circular pattern as is represented in Fig. 5.

The coefficients of the window mask are stored inside the architecture in a memory bank that is able to shift data from one element to its neighbor. A shift register bank is distributed on internal registers of the processing elements to delay the mask coefficients.

In a similar way to the one used to read the input data, the memory containing the coefficients of the window mask of a window operator is read in a column-based scan. Fig.6 shows the reading process of the mask coefficients as time progresses. The coefficients are read sequentially and their values are transmitted to different window processors when an image is being processed.

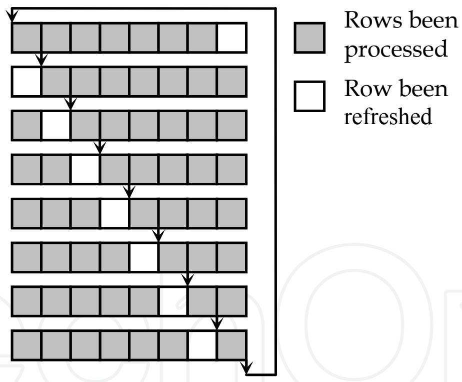


Fig. 5. Circular pipeline in the buffer memory

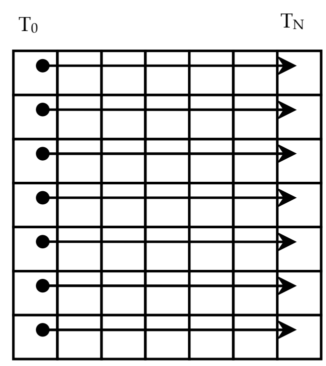


Fig. 6. Reading pattern for window mask

The reading process of the window mask coefficients and input image pixels requires a synchronization mechanism to match the operations sequence.
For simplicity the control unit for the systolic array has not been show in Fig. 2. This module is in charge of generating all the control and synchronization signals for the elements of the architecture.
The control unit synchronizes external memory, input and output buffers banks, and systolic array computations. The control unit indicates which processors execute an operation and when a result must be sent to the output storage elements. The control unit has been decomposed into local and simpler control circuits which are synchronized through a restricted set of signals. Therefore several distributed control sub-units exist in the systolic array to manage data flow in the PEs, to generate output memory addresses, and systolic array computations.

3.2 Processing element

Each PE has been specially designed to support the operations involved in most window-based operators in image processing: Multiplication, addition, subtraction, accumulation, maximum, minimum, and absolute value.
One processing element comprises one arithmetic processor (ALU) and a local reduction module (Accumulator) and can be configured by a control word selected by the user as can be observed in Fig. 7.
The PE has two operational inputs, incoming pixels from the input image (p) and coefficients from the window mask (w). Each PE has two output signals, the partial result of the window operation and a delayed value of a window coefficient (w_d) that is transmitted to its neighbor PE. For every clock cycle, each PE executes three different operations in parallel:

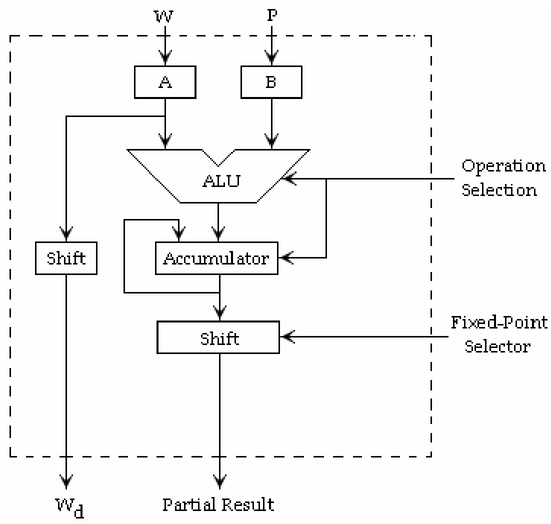


Fig. 7. Processing element implementation

- Computes the pixel by pixel value to be passed to the next computation cycle
- Integrates the contents of the outputs registers calculated at the previous clock cycle, with the new value produced in the arithmetic processor (ALU).
- Reads a new mask coefficient and stores it into the register. Then, transmits the previous coefficient to the next PE.

When the systolic pipeline is full a window output is obtained every cycle providing a throughput of 1.

4. Extension to the architecture for motion computation

In order to provide more capacity to the architecture and to turn it into a real platform, the basic structure has been modified to support the Motion Estimation (ME) algorithm. To implement ME in coding image applications, the most popular and widely used method, is the Full Search Block-Matching Algorithm (FBMA) (Gui-guang & Bao-long, 2004).
The FBMA divides the image in squared blocks, macro-block (MB), and compares each block in the current frame (reference block) with those within a reduced area of the previous frame (search area) looking for the best match (Kuhn, 1999). The matching position relative to the original position is described by a motion vector, as has been illustrated in Fig. 8.
$I_k(x, y)$ is defined as the pixel intensity at location (x, y) in the k-th frame and $I_{k-1}(x, y)$ is the pixel intensity at location (x, y) at the k-1-th frame. For FBMA motion estimation, $I_{k-1}(x, y)$, represents usually a pixel located in the search area of the size $R^2 = R_x \times R_y$ pixel of the reference frame and $I_k(x, y)$ belongs to the current frame. The block size is defined as $N^2 = N \times N$ pixel. Each individual search position of a search scheme is defined by $\overline{CMV} = (dx, dy)$.
The matching procedure is made by determining the optimum of the selected cost function, usually Sum of Absolute Differences (SAD), between the blocks (Saponara & Fanucci, 2004). The SAD is defined as:

$$SAD(dx,dy) = \sum_{m=x}^{x+N-1} \sum_{n=y}^{y+N-1} \left| I_k(m,n) - I_{k-1}(m+dx,n+dy) \right| \qquad (2)$$

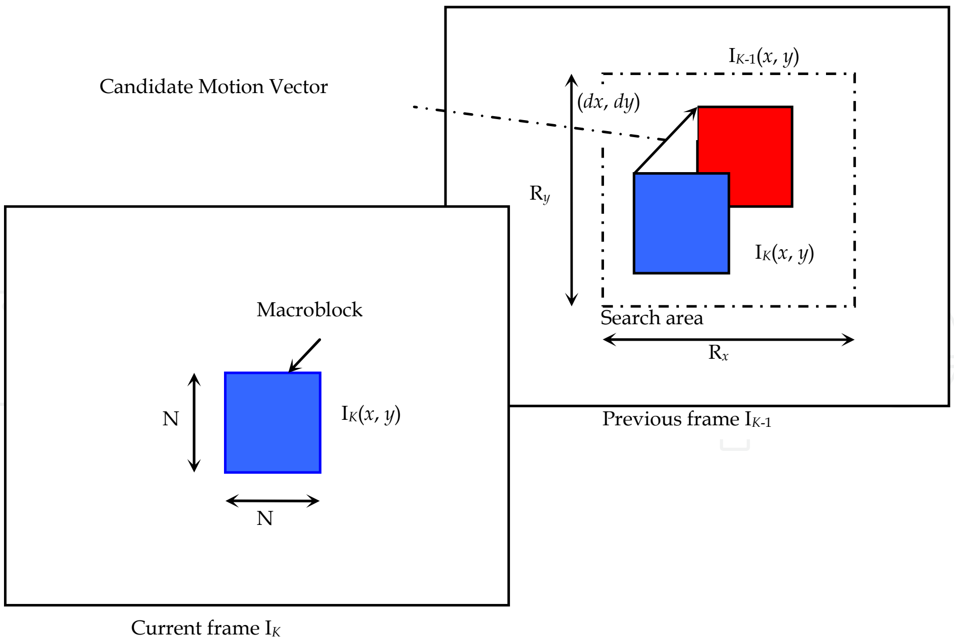


Fig. 8. Block-matching for motion estimation

$$\overline{MV} = (MV_x, MV_y) = \min_{(dx,dy)\in R^2} SAD(dx, dy) \quad (3)$$

The motion vector $\overline{MV}$ represents the displacement of the best block with the best result for the distance criterion, after the search procedure is finished.
Due to the nature of Equation (2) the FBMA can be formulated as a window-based operator, though some aspects must be considered:

- The coefficients of the window mask are variable and new windows are extracted from the first image to constitute the reference block. Once the processing in the search area has been completed, the window mask must be replaced with a new one, and the processing goes on the same way until all data is processed.
- The different windows to be correlated are extracted in a column-based order from the search area to exploit data overlapping and sharing. The pixels are broadcasted to all the processors to work concurrently.

Based on these characteristics, the processing block has been modified to support SAD operation required for FBMA.
When the SAD value is processed, data is available in a row format therefore when blocks are processed vertically; previous read data in the search area are overlapped for two block search as shown in Fig. 9.
In order to reuse the image pixel available, the PE has been modified to work with a double ALU scheme to process two blocks in parallel. The final structure is observed in Fig. 10.

5. Performance discussion

In this chapter some representative algorithms based on windows-operators convolution, filtering, matrix multiplication, pyramid decomposition and morphological operators have been presented in order to validate the correct functionality of the proposed architecture and its generalization as a hardware platform. The technical data presented for each version

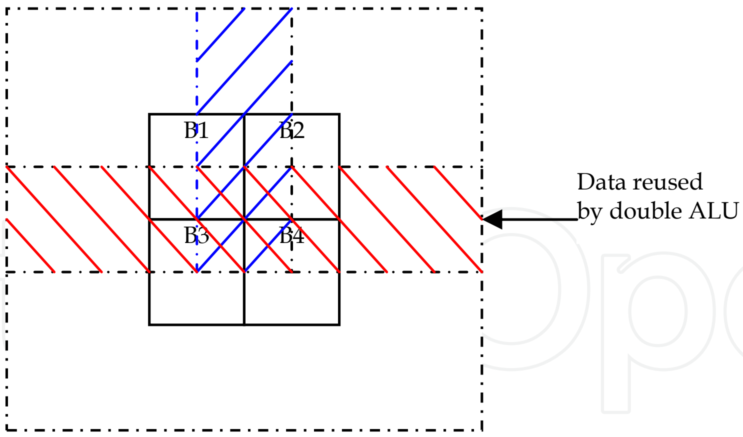


Fig. 9. Data overlapped between search areas in the horizontal and vertical direction for ME

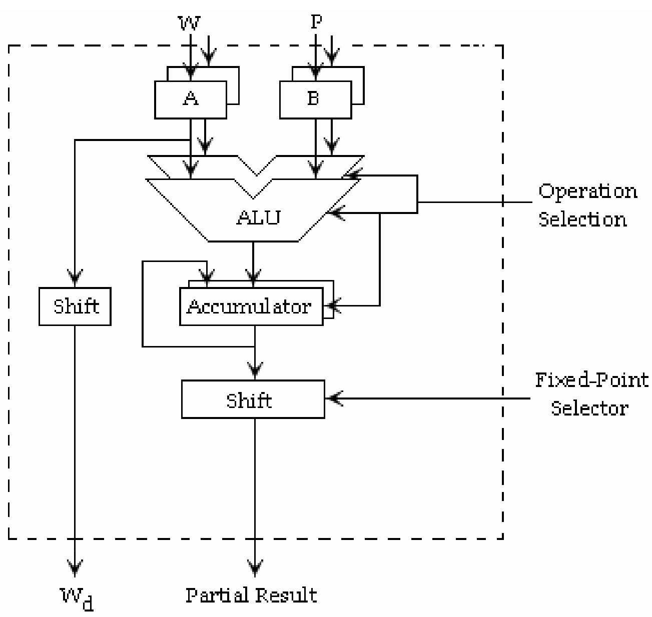


Fig. 10. PE modified structure to support ME algorithm

of the architecture constitute a measurement of its performance. The three main parameters considered are the speed, the throughput and the power consumption. Table 1 summarizes the results obtained for this set of algorithms.

Application	Number of Slices	Clock Frequency	Power Consumption
Convolution	11,969 out of 19200	66 MHz	2.017 W
Filtering	11,969 out of 19200	66 MHz	2.017 W
Matrix multiplication	11,969 out of 19200	66 MHz	2.017 W
Gaussian pyramid	11,969 out of 19200	66 MHz	2.017 W
Erosion	12,114 out of 19200	66 MHz	2.4 W
Dilation	12,074 out of 19200	66 MHz	2.017 W

Table 1. Summary of the architecture performance

From this table it can be observed little variations in the area occupied according to the algorithm being performed. These changes are due to the configuration selected for the PEs and the scalar operation being performed. However the performance and power consumption practically remain the same.
In order to establish the advantages of the presented architecture, the results obtained in Table 1 needs to be compared with previous implementations of image processing architectures; even though most performance metrics are rarely reported for architectures and systems in literature. This lack of standard metrics for comparison makes difficult to determine the advantages of a given system.
(DeHon, 2000) proposed a model to compute the hardware resource utilization in a system considering the fabrication technology. This model provides a standard metric that allows doing a fair comparison between systems measuring the silicon area in feature size units rather than in absolute units.
The silicon area required by the architecture is computed in terms of the feature size in λ. Considering data for the XCV2000E device and the results obtained by (DeHon, 2000) and (Torres-Huitzil, 2003) it is possible to present a comparison with previous architectures. For this purpose the execution time, given in milliseconds, and the silicon area occupied are considered as main metrics. The assessments were made considering that the systems deal with the same algorithm and they use the same image size. Table 2 presents the technical details for the chosen architectures.

System	Architecture	Application	Image Size	Timing	Silicon Area
(Rosas, 2005)	SIMD FPGA-based	3×3 Filtering	640×480	23.04 ms	Not reported
(Vega-Rodriguez, 2004)	FPGA-based	3×3 Filtering	640×480	868.51 ms	322 $G\lambda^2$
(Torres-Huitzil, 2003)	Systolic FPGA-based	7×7 Generic Window-based Image operator	640×480	9.7 ms	15 $G\lambda^2$
(Vega-Rodriguez, 2002)	Systolic FPGA-based	7×7 Median Filter	640×480	998.20 ms	1.41 $G\lambda^2$
(Herrmann , 2004)	Von Newman	3×3 Generic Convolution	640×480	2863 ms	N/A
Proposed Architecture	Systolic	7×7 Generic Window-based operators	640×480	5 ms	26.7 $G\lambda^2$

Table 2. Performance for different architectures

In summary, the proposed architecture provides a throughput of 5.9 GOPs for this set of algorithms on a chip area of 26.7 $G\lambda^2$ with an estimated power consumption of 2.4 W running at 66 MHz clock frequency, which is a good compromise in area and power consumption for the attained performance. From these results it can be shown that it is possible to achieve real-time performance for applications based on windows operators. Furthermore, the capacity of generalization for the proposed schema has been established.

6. Implementation and results

For test and validation purposes, a RC1000 board from Celoxica that supports a XCV2000E XILINX Virtex-E FPGA with up to 2 million system gates, 640×480 gray-level images and sequences were used. Even though window masks of different size can be employed, only results for 7×7 are presented. Technical details for the implementation are shown in Table 3. The hardware resource utilization for the complete architecture is about 63% of total logic available in the FPGA.When the double ALU scheme is activated the Peak performance grows up to 9 GOPs.

Element	Specification
Virtex-E	XCV2000E
FPGA technology	0.18 µm 6-layer metal process
Number of PEs	49
Off-chip memory data buses	21 bit-address, 32 bit data
Internal data buses for ALUs	8 bits for fixed-point operations
Number of Block RAMs:	13 out of 160
Number of Slices	12,114 out of 19200
Number 4 input LUTs	19,163 out of 38,400
Number of Flip Flops	4,613 out of 38,400
Overall % occupancy	63%
Clock frequency	66 MHz
Estimated Power Consumption	2.4 W
Peak performance	~5.9 GOPs

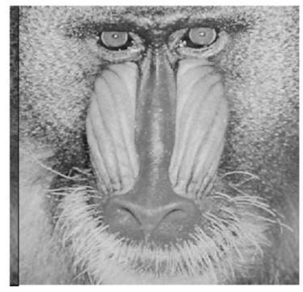
Table 3. Technical data for the entire architecture

In order to prove the architecture versatility several window-based algorithms have been tested in the FPGA board, filtering, erosion, dilation, Gaussian pyramid, and matrix by matrix multiplication. Some images examples obtained during experiments are shown in Fig. 11.
Table 4 summarizes the technical details obtained for the motion estimation algorithm.

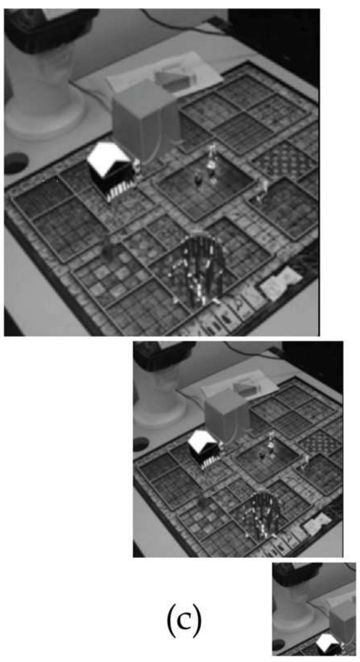
7. Conclusions and future work

In this paper a versatile, modular and scalable platform for test and implementation of low-level image processing algorithms under real-time constraints was presented.
The architecture consists of a programmable array of processors organized in a systolic approach. The implementation can achieve a processing rate of near 5.9 GOPs with a 66MHz clock frequency for the window processing. The performance increased to 9 GOPs for the motion estimation architecture extension. The high-performance and compact hardware architecture opens new and practical possibilities to mobile machine vision systems where size and power consumption are hard constraints to overcome.

(a)



(b)



(c)

$$A = \begin{pmatrix} 1 & 2 & 6 & 5 & 8 & 9 & 9 \\ 5 & 4 & 2 & 5 & 4 & 1 & 2 \\ 5 & 9 & 3 & 2 & 1 & 4 & 7 \\ 5 & 4 & 5 & 7 & 8 & 9 & 6 \\ 3 & 2 & 1 & 6 & 7 & 5 & 3 \\ 11 & 3 & 2 & 7 & 3 & 5 & 9 \\ 12 & 10 & 5 & 3 & 9 & 7 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 7 & 2 & 8 & 5 & 1 & 19 & 9 \\ 5 & 41 & 2 & 5 & 9 & 10 & 2 \\ 7 & 22 & 3 & 2 & 6 & 4 & 8 \\ 5 & 4 & 5 & 17 & 8 & 19 & 6 \\ 3 & 12 & 5 & 6 & 14 & 5 & 1 \\ 1 & 3 & 2 & 27 & 13 & 5 & 9 \\ 2 & 9 & 5 & 30 & 9 & 17 & 2 \end{pmatrix}$$

$$A \times B = \begin{pmatrix} 135 & 440 & 158 & 673 & 405 & 396 & 198 \\ 111 & 307 & 111 & 245 & 180 & 297 & 116 \\ 132 & 540 & 125 & 434 & 249 & 379 & 150 \\ 170 & 489 & 186 & 645 & 410 & 475 & 236 \\ 100 & 260 & 121 & 396 & 265 & 306 & 133 \\ 173 & 349 & 205 & 606 & 294 & 573 & 229 \\ 222 & 703 & 215 & 474 & 391 & 519 & 262 \end{pmatrix}$$

(d)

Fig. 11. Window-based algorithms implemented: (a) Filtering, (b) Morphologic Operators, (c) 2 level Gaussian pyramid, (d) Matrix Multiplication.

The configurable architecture developed can be use to support different algorithms based on windows processing such as generic convolution, filtering, gray-level image morphology, matrix multiplication and Gaussian pyramid. In addition the architecture provides support to the algorithm of motion estimation that is one of the most computationally demanding in video applications achieving bandwidth efficiency for both transmission and storage with reduced power consumption.

The programmability of the proposed architecture provides the advantage of being flexible enough to be adapted to other algorithms such as template matching and stereo disparity computation, among others. In this sense, considering the broad range of algorithms that can be implemented in the architecture, it is a convenient platform to develop and accelerate image processing applications under real-time constraints.

The platform has proven to be capable of handling a large amount of data with low area utilization, to benefit from parallelism as well as to attain a higher data transfer using a reduced bus bandwidth. The main focus has been placed on communication, and the possibility of processes chaining. Image buffers and Router elements allow cascade connection of several processing stages.

Element	Specification
Virtex-E	XCV2000E
FPGA technology	0.18 μm 6-layer metal process
Number of PEs	49
Off-chip memory data buses	21 bit-address, 32 bit data
Internal data buses for ALUs	8 bits for fixed-point operations
Number of Block RAMs:	18 out of 160
Number of Slices	12,100 out of 19200
Number 4 input LUTs	5,600 out of 38,400
Number of Flip Flops	7,742 out of 38,400
Overall % occupancy	65%
Clock frequency	66 MHz
Estimated Power Consumption	3 W
Peak performance	~9 GOPs

Table 4. Technical data for ME algorithm

The performance comparison with other existing architectures confirms the promising advantages of the proposed FPGA-based systolic architecture over other conventional approaches. Its performance has been evaluated for the previous window-based algorithms with excellent results that validate the proposed high-performance architectural model.

Furthermore, the design can be extended using dynamic reconfiguration techniques at high level, that is, the processor array could be reconfigured for different parts of a high level image processing chain, reusing the existing Routing, I/O Buffer and Data Flow Control structures. Dynamic reconfiguration allows modifying an application architecture at run time, therefore the platform capacities can be extended beyond what has been presented in this chapter without large increase in FPGA resource requirements. Selectively modification of the system operation at run time would allow the architecture to execute a sequence of different window-based operators to processes chaining, reusing the same hardware resources which implies a reduction in area occupancy and power consumption. This approach is currently been explored in order to determine its capacities.

8. References

Ballard, D. H. & Brown, C. M. (1982). *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, USA

Benkrid, K., et all. (2001). High Level Programming for FPGA based Image and Video Processing using Hardware Skeletons, *Proceedings of the Symposium on Field-Programmable Custom Computing Machines*, pp. 219-226, ISBN: 0-7695-2667-5, April 2001, IEEE Computer Society, Washington, DC

Bouridane, A., et al (1999). A high level FPGA-based abstract machine for image processing, *Journal of Systems Architecture*, Vol. 45, No. 10, (April 1999), pp. 809-824, ISSN: 1383-7621

DeHon, A. (2000). The Density Advantage of Configurable Computing, *IEEE Computer*, Vol. 33, No. 4, (April 2000), pp. 41-49, ISSN: 0018-9162

Gui-guang, D. & Bao-long, G. (2004). Motion Vector Estimation Using Line-Square Search BlockMatching Algorithm for Video Sequences, *Journal on Applied Signal Processing*, Vol. 2004, No. 11, (January 2004), pp. 1750-1756, ISSN: 1110-8657

Herrmann, C. & Langhammer, T. (2004). *Automatic Staging for Image Processing*, Technical Report, Fakultät für Mathematik und Informatik, Universität Passau, Germany

Kuhn, P. (1999). *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation,* Kluwer Academic Publishers, ISBN-13: 978-0792385165, USA

Li, D., Jiang, L. & Kunieda, H. (1999). Design optimization of VLSI array processor architecture for window image processing, *IEICE Transactions on Fundamentals*, Vol. E-82-A, No. 8, (August 1999), pp. 1474-1484, ISSN: 0916-8508

Managuli, R., et al. (2000). Mapping of two dimensional convolution on very long instruction word media processors for real-time performance, *Journal of electronic Imaging*, Vol. 9, No. 3, (April 2000), pp. 327–35, ISBN: 10.1117/1.482755

Reuver, D. & Klar, H. A Configurable Convolution Chip with Programmable Coefficients, *IEEE Journal of Solid State Circuits*, Vol. 27, No. 7, (July 1992), pp. 1121-1123, ISSN: 0018-9200

Rosas, R. L., De Luca, A. & Santillan, F. B. (2005), SIMD architecture for image segmentation using Sobel operators implemented in FPGA technology, *Proceedings of the 2nd International Conference on Electrical and Electronics Engineering*, pp. 77-80, ISBN: 0-7803-9230-2, September 2005, IEEE Computer Society, Mexico

Saponara, S. & Fanucci, L. Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems, *IEE Proceedings - Computers and Digital Techniques*, Vol. 151, No. 1, (January 2004), pp. 51-59, ISSN: 1350-2387

Torres-Huitzil C. (2003). *Reconfigurable Computer Vision System for Real-time Applications*, Ph.D. Thesis, INAOE, Mexico

Torres-Huitzil, C. & Arias-Estrada, M. (2005), FPGA-Based Configurable Systolic Architecture for Window-Based Image Processing, *EURASIP Journal on Applied Signal Processing*, Vol. 2005, No. 7, (January 2005), pp. 1024-1034, ISSN:1110-8657

Umbaugh, S.E. (1998). *Computer Vision and Image processing - a practical approach using CVIPtools*, Prentice Hall, ISBN-13: 978-0132645997, USA

Vega-Rodriguez, et al. (2002). An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems, *Proceedings of the 10th Mediterranean Conference on Control and Automation*, pp. 131-136, July 2002, Lisbon, Portugal

Vega-Rodriguez, et al. (2004). An optimized architecture for implementing image convolution with reconfigurable hardware, *Proceedings of the World Automation Congress*, Vol. 16, pp. 131-136, ISBN: 1-889335-21-5, June-July 2004, Spain

Villasenor, J. & Hutchings, B., The flexibility of configurable computing, *IEEE Signal Processing Magazine*, Vol. 15, No. 5, (September 1998), pp. 67–84, ISSN: 1053-5888k

Image Processing

Edited by Yung-Sheng Chen

ISBN 978-953-307-026-1

Hard cover, 516 pages

Publisher InTech

Published online 01, December, 2009

Published in print edition December, 2009

There are six sections in this book. The first section presents basic image processing techniques, such as image acquisition, storage, retrieval, transformation, filtering, and parallel computing. Then, some applications, such as road sign recognition, air quality monitoring, remote sensed image analysis, and diagnosis of industrial parts are considered. Subsequently, the application of image processing for the special eye examination and a newly three-dimensional digital camera are introduced. On the other hand, the section of medical imaging will show the applications of nuclear imaging, ultrasound imaging, and biology. The section of neural fuzzy presents the topics of image recognition, self-learning, image restoration, as well as evolutionary. The final section will show how to implement the hardware design based on the SoC or FPGA to accelerate image processing.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821