8-13

MVA2007 IAPR Conference on Machine Vision Applications, May 16-18, 2007, Tokyo, JAPAN

A Configurable Window-Based Processing Element for Image Processing on Smart Cameras

Fabio DIAS, François BERRY, Jocelyn SEROT, François MARMOITON
LASMEA - Laboratoire des Sciences et Matériaux pour l'Electronique et d'Automatique
Université Blaise Pascal - 24 avenue des Landais - 63177 Aubière - FRANCE
{dias, berry, jserot, marmoito}@lasmea.univ-bpclermont.fr

Abstract

Image processing in real-time is known as a high processing-power consuming task. Smart cameras deal with this problem by adding embedded processing resources to the camera architecture. This way, early low-level vision tasks can be done internally, before transmission to the host system and helping to prevent from communication bottlenecks. This paper presents a configurable window-based processing element, tailored for implementation on a FPGA device. Window-based operations are often required for image filtering, feature detection, tracking, etc. In this work, some particular points of these applications are analyzed, and it is shown that a general computational model can be extracted. This model is applied to conceive a processing element, able to perform several window-based operations. A hardware implementation of the configurable window-based operator is proposed, and a smart camera platform, able to host such processing element, is presented and detailed.

Keywords: *Smart camera, embedded processing, SIMD processing, reconfigurable architecture, FPGA-based system.*

1 Introduction

Real-time image processing is a challenging domain for research and industry. Big amounts of data and intensive repetitive calculations are two major problems to deal with when conceiving a system that must process images at video rate. Even with recent outstanding advances in microelectronics, standard PC architectures are frequently unable to deliver required performances for such applications.

Cameras with embedded processing resources, also known as smart cameras, may be a solution to deal with such artificial vision constraints. The approach consists of executing early vision tasks by the camera itself, before data transmission to the host system. Early vision tasks therefore allows to reduce data amounts to be transmitted, avoiding communication bottlenecks and reducing overheads. For example, in a tracking application, the camera can transmit only the image portion containing the tracked pattern or object, strongly reducing the communication flow (face tracking application for mobile videophones for instance).

On the other hand, transmitted data can be more pertinent than raw pixel flow coming from the image sensor. The enhanced pertinence would help the host system to simplify some high-level tasks. For example, an embedded corner and edge detector allows to send to the host system only a list of characteristic points, and small image samples containing these points. This strategy allows to reduce the communication flow, and saves the host system from executing a feature detector over the whole image. If the full image is needed for a given operation, it can be transmitted in low resolution, through embedded down-sampling.

Smart cameras [2] and smart camera networks [1] [4] are both subjects of growing interest. Several domains and applications are concerned, among them video-surveillance, robotics, quality control and intelligent vehicles. Processing resources may be added to the camera hardware in form of FPGA devices, embedded processors, DSP's, etc. However, addition of these devices makes the implementation tasks more complex, as the heterogeneous nature of the system requires co-design techniques and tools. The work presented here is a part of an approach towards a design methodology for heterogeneous smart camera platforms.

This methodology relies on a programming tool able to generate a suitable hardware architecture for a given application model. The generated architecture is based on a set of configurable processing elements, which must be able to deal with scalar data, vectors and also matrices (image samples), which is the most natural data format when dealing with images. A managing module is responsible for task sharing, data deployment and synchronization of different processes.

Early vision tasks frequently involve neighborhood operators, combining several adjacent pixels to produce a single result. Examples are convolution (spatial filters, wavelets), correlation estimation (SAD, SSD) and morphological transformations (dilatation and erosion). These operations imply a great number of memory accesses, and repetitive computations over the whole image. Being time and processing expensive, the optimization of these processes is essential to achieve temporal performances imposed by the real-time constraint. Parallelism must be massively exploited, and memory access redundancy must be reduced.

This paper proposes a window-based processing element, tailored for implementation on a FPGA device in a smart camera platform. The central idea is to detect computational similarities among different window-based operators, and to exploit these common features in order to conceive a configurable processing element.

This paper is organized as follows: in the next section, some window-based operations frequently employed in image processing are analyzed, and a common computational structure is extracted. In section 3, a functional architecture for a configurable window-based processing element is proposed. Thus, in section 4, the smart camera research platform is briefly described, followed by some conclusions and perspectives for future work in the last section.

2 Window-based operations

2.1 General computational model

Window-based operations are often applied for low-level image processing. Median, mean and gaussian filtering, correlation estimation for feature matching, image difference for background extraction, all these operations are window-based or can be expressed as it. Thus, a processing element able to deal with this class of applications may be of particular interest for a smart camera. In order to design such a processing element, some frequently employed neighborhood operators are reviewed in this section. The goal is to show that some basic features are shared by different operators, and that a general computational model for window-based processing can be extracted. The interest of such model is to inspire a generic structure for low-level image processing, analog to an ALU (Arithmetic Logic Unit) on a processor architecture.

We consider that a window-based operation can be performed following a model that gets two image samples as inputs and may produce two different outputs: an image sample or a scalar value. In this way, we argue that a window-based operation is a composition of 3 functions such as:

$$x = \mathbf{F_R}(\mathbf{F_M}(\mathbf{F_D}(A_{(i,j)}, B_{(i,j)}))) \tag{1}$$

$$Q_{(i,j)} = \mathbf{F_M}(\mathbf{F_D}(A_{(i,j)}, B_{(i,j)})) \tag{2}$$

where $A_{(i,j)}$ and $B_{(i,j)}$ are the input image operands, $Q_{(i,j)}$ is an image result and x is a scalar result.

• The first function $\mathbf{F_D} : (\mathbb{Z}, \mathbb{Z}) \rightarrow \mathbb{Z}$ applies independently for each pair of elements from the image operands $A_{(n\times n)}$ and $B_{(n\times n)}$. The result from $\mathbf{F_D}$ function is an image $R_{(n\times n)}$ where:

$$R_{(i,j)} = \mathbf{F_D}(A_{(i,j)}, B_{(i,j)}) \tag{3}$$

Typically, the $\mathbf{F_D}$ function is a simple arithmetic or logic operation.

• The second function $\mathbf{F_M} : \mathbb{Z} \rightarrow \mathbb{Z}$ has only one input image operand. This function is applied independently for each element of $R_{(n\times n)}$, producing the image result $Q_{(n\times n)}$ such as:

$$Q_{(i,j)} = \mathbf{F_M}(R_{(i,j)}) \tag{4}$$

Classically, $\mathbf{F_M}$ is normalization, absolute value, thresholding, ...

• The last operation is a reduction function $\mathbf{F_R} : (\mathbb{Z}^2) \rightarrow \mathbb{Z}$, applied on all n^2 elements of the matrix $Q_{(n\times n)}$, and producing a scalar result x:

$$x = \mathbf{F_R}(Q_{(i,j)}) \tag{5}$$

Our goal is to show that equations 1 and 2 can describe a great number of window-based operations, depending on the choice for functions $\mathbf{F_R}$, $\mathbf{F_M}$ and $\mathbf{F_D}$. To that, some examples among the most frequently employed window-based operations are analyzed below.

2.2 Examples of low-level processing

Convolution is one of the most employed operations for low-level processing. Several different results can be obtained, depending on the employed mask. The general mathematical formula is given below, where A is a $(n \times n)$ window around pixel (x, y) from the input image, and B is the constant $(n \times n)$ convolution mask:

$$\mathbf{Conv}(x, y) = \frac{1}{k} \sum_{(i,j)} \{A(i, j) \times B(i, j)\} \tag{6}$$

Constant k is a normalization factor, and depends on the nature of the mask. The same calculation must be performed for each pixel on the input image, producing one pixel of the resulting image, except for border areas. The convolution computation for pixel (x, y) can be expressed as:

$$R(i, j) = A(i, j) \times B(i, j) \tag{7}$$

$$Q(i, j) = R(i, j)/k \tag{8}$$

$$Conv(x, y) = \sum_{(i,j)} Q(i, j) \tag{9}$$

Thus, equation 1 can express a convolution when $\mathbf{F_D}$ is a multiplication, $\mathbf{F_M}$ is a division by k (or multiplication by 1/k, or bit shifts if k is a power of 2) and $\mathbf{F_R}$ is a sum.

Table 1: $\mathbf{F_D}$, $\mathbf{F_M}$ and $\mathbf{F_R}$ for different window-based operations

Operations	$\mathbf{F_D}$	$\mathbf{F_M}$	$\mathbf{F_R}$
Convolution	A × B	R/k	$\sum Q$
SAD	A - B	$\lvert R \rvert$	$\sum Q$
SSD	A - B	R^2	$\sum Q$
Mean filter	A × 1	R/n^2	$\sum Q$
Max filter	A × 1	R	$Max\ Q$
Image Difference	A - B	$\lvert R \rvert$	-
Binary ImD	A - B	$thold(R)$	-
Eroded B_ImD	A - B	$thold(R)$	AND Q
Binary Dilation	A and B	R	OR Q
Binary Erosion	A or !B	R	AND Q
Greyscale Dilation	A + B	R	$Max\ Q$
Greyscale Erosion	A - B	R	$Min\ Q$

Another example of window-based processing is **correlation** computation. This technique is currently employed to perform feature matching, detecting the presence of a given pattern in an image (an object or part of it for example). The pattern is defined as a $(n \times n)$ sample, which will be compared with each portion of the input image in order to detect its presence and location. One of the most used correlation functions is the sum of absolute differences (SAD), as described in equation 10 where A is an image sample around pixel (x, y), and B is the searched pattern:

$$\mathbf{SAD}(x, y) = \sum_{(i,j)} |A(i, j) - B(i, j)| \tag{10}$$

Equation 10 shows that the SAD computation can also be decomposed into three functions ($\mathbf{F_D}$ is a substraction, $\mathbf{F_M}$ is an absolute value function and $\mathbf{F_R}$ is a sum).

Other window-based algorithms that can be taken as examples are **image difference** (**ImD** eq.11) or **binarized image difference** (**B_ImD** eq. 12):

$$\mathbf{ImD}(i, j) = |A(i, j) - B(i, j)| \tag{11}$$

$$\mathbf{B\_ImD}(i, j) = Threshold[A(i, j) - B(i, j)] \tag{12}$$

These operations are currently employed for motion detection, with A and B being parts of two consecutive images from a sequence [3]. $\mathbf{F_D}$ is a subtraction for both operations, $\mathbf{F_M}$ is an absolute value for the first and a thresholding function for the second. However, outputs of these functions are matrices, and not scalars. So, the reduction function $\mathbf{F_R}$ does not need to be performed.

Morphological transformations can also be considered. In this case, operand B is the structuring element. These operations and other examples are shown in table 1.

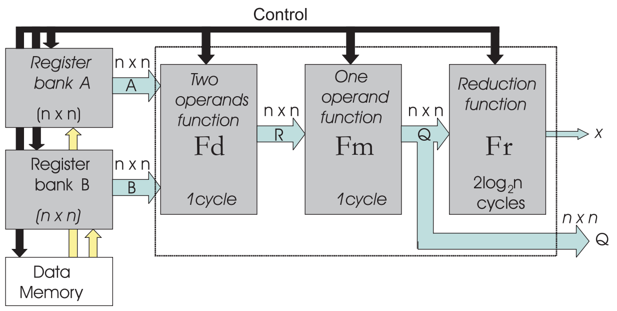
Control
Register bank A (n x n)
Register bank B (n x n)
Data Memory
n x n
A
B
Two operands function Fd 1cycle
R
One operand function Fm 1cycle
Q
Reduction function Fr $2\log_2 n$ cycles
x
n x n
Q

Figure 1: Proposed global architecture for the processing element.

3 Configurable Processing Element

As said in section 1, window-based processing requires a big number of elementary operations and memory accesses. For a convolution by a $(n \times n)$ mask for example, one will need $(2 \times n^2)$ memory readings (data and mask loading), (n^2) multiplications, $(n^2 - 1)$ additions, one division and one memory writing (to store result). These operations must be repeated $(L - n + 1) \times (W - n + 1)$ times, where (L, W) is the image size. If no optimizations are done, the same pixel can be read up to n^2 times. To achieve satisfactory performances, memory reading redundancy must be reduced, and the intensive repetitive calculations must be parallelized.

The classic approach for this kind of operation is to employ line buffers, storing $n - 1$ lines from the input image into FIFO structures, and receiving data directly from the sensor's output pixel flow. Once the input pipeline is full, processing begins producing one result for each data received, eliminating data redundancy. However, line buffers requires a lot of hardware resources, specially if the input image is big. Moreover, if the input image size changes, the system architecture must also be changed (FIFO depth). In this section, a flexible solution based on the computational model presented above is discussed. Other solutions can be found in [6] and [7].

First considerations are about parallelism. As seen in equations 3 and 4, functions $\mathbf{F_D}$ and $\mathbf{F_M}$ are applied independently for each input data element. It means that there is an inherent exploitable data parallelism, with several logic/arithmetic operations being executed simultaneously in a SIMD structure.

A second source of potential parallelism is in the computational model itself (eq. 1). Three functions are executed, always in the same order, and receiving the result of the previous function as input data. A pipeline structure can be considered as a natural architectural model to fit this kind of process. So, to exploit both kinds of parallelism (data and task), we propose a processing element consisting on a three levels pipeline, with SIMD units in first two levels (figs. 1 and 2).

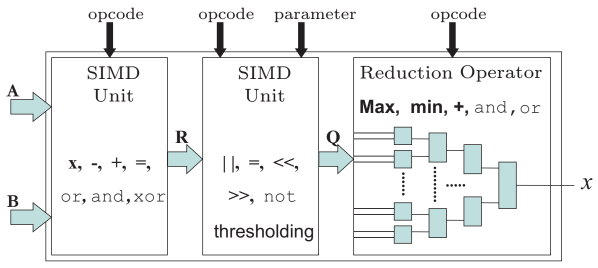

Figure 2: Configurable window-based processor.

First level SIMD unit ($\mathbf{F_D}$) is able to perform simultaneously several logic or arithmetic operations, as logic `AND`, `OR`, `XOR`, multiplication, addition, subtraction, and equality (multiplication by 1). Latency of this stage is one clock cycle.

Second level SIMD unit ($\mathbf{F_M}$) performs operations as absolute value, equality, thresholding, bit shifts and logic `NOT`. A parameter input defines the threshold or the number of bit shifts to perform. Multiplication and division are not available in the operation set. A multiplication would be necessary for **SSD**, but this function is currently replaced by **SAD**, which is less computational expensive and has the same qualitative properties [5]. If a division by a non-power of 2 is needed, it's better to divide the scalar result x after the reduction function, because several dividers in parallel would be expensive in hardware terms. Latency of this stage is one clock cycle.

The third level is the reduction function ($\mathbf{F_R}$). Logic `AND`, `OR`, addition, max or min operations are applied over input data, combining two elements per two in a dyadic tree structure. If the input matrix is $(n \times n)$, $2 \times \log_2 n$ stages are needed. This is the latency for the third level in clock cycles. Total latency of the pipeline is $2 + 2 \times \log_2 n$.

For some operations like image difference, the reduction function is not applied. For this reason, it is possible to output the matrix Q before the end of the pipeline.

Input data coming from the image sensor is stored into RAM memories. Then, input operands are loaded from memory into two $n \times n$ register banks, and all $2 \times n^2$ registers can be accessed simultaneously by the first SIMD unit. If necessary, the registers structure can be adapted to fit a given application. For example, the input register bank B can be replaced by (or multiplexed with) the Q output feedback, working as an accumulator on a classic ALU.

A managing module is responsible for input registers loading and for configuring each stage (opcode setting) to perform a given function. The processing element is able to execute different operations according to the control settings, functioning as an image processing ALU. If the application only needs one or two kinds of operations, a "light" version of the processing element can be generated, for spatial optimization purposes.

Registers loading strategy is of particular importance to reduce memory access redundancy. As said above, the same pixel can be used in up to n^2 computations. A simple way to optimize registers loading is being able to shift the input register matrix. If a n^2 window is loaded for a computation, and the next one is performed over the same window translated of one pixel to the right, our input register has $(n-1)n$ values which can be reused. In this case, the input matrix is shifted of one column, and only the n new elements are loaded. The memory readings number is therefore divided by n.

If operand B is constant (as frequently), it is loaded only once. If not, a dual-ported memory (or two separate memories) should be used to allow both register banks being loaded simultaneously.

A last consideration is about the choice of element's dimension (n). As registers loading has a lot of influence in the global performance, choosing a big n is useless if the memory structure is unable to load the input registers fast enough. Speedups obtained through parallelization will be lost due to memory access overhead. Memory type, structure, cadence, and word size must be chosen in order to ensure that the processing element will be able to deliver its optimal performance for a given n. To that, register loading frequency must be n times greater than the processing pipeline functioning frequency, in order to avoid that the processing element remains idle.

The window-based processing element proposed here was implemented on a smart camera research platform. Its hardware architecture is briefly described in next section.

4 Smart Camera Platform

The smart camera platform (fig. 3) is based on a CMOS image sensor and an ALTERA Stratix FPGA device. The purpose of such platform is to perform early vision tasks in real-time, before data transmission to the external host system.

The Stratix EP1S60 FPGA device plays the central role in the system, being responsible to interconnect all other hardware devices. Surrounding it, 10Mb (5x 1MWords) of SRAM and 64Mb of SDRAM are available for image and data storage.

The choice of a CMOS image sensor is justified by its random addressing capabilities. This feature is extremely important when dealing with applications as feature tracking, where WOI (window of interest) acquisition is needed. It allows to have a high-resolution sensor (4 Mega-pixels), but not acquiring in full resolution all the time, what would be prohibitive face to time constraints of some applications. Random addressing makes possible to have high resolution images or high frame rates with the same sensor device, and

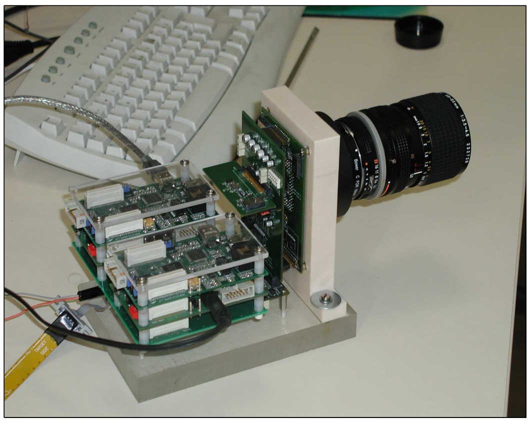

Figure 3: Smart Camera research platform.

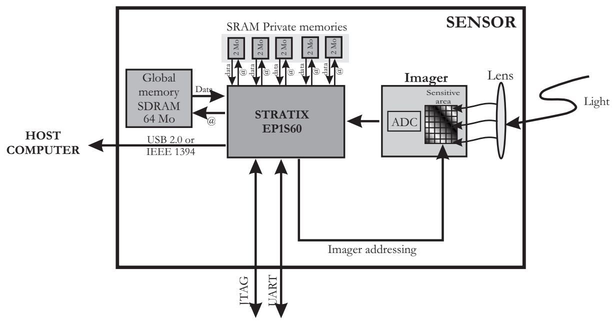

Figure 4: Hardware architecture synoptic scheme.

according to the application needing.

One of the main features of this hardware platform is its modularity and flexibility. Different hardware devices are integrated in different boards (except for memories, which share the same board as the FPGA). These boards are interconnected together, and can be replaced to allow the evolution of the platform. Thanks to this feature this smart camera platform has frequently evolved. The actual configuration features a 4Mpix LUPA-4000 image sensor, from Fill Factory/Cypress Semiconductors, and a Firewire Interface.

The LUPA-4000 image sensor can acquire up to 66Mpixels per second, and full resolution format is 2048 x 2048, each pixel coded in 10 bits. Image capture is done in global shutter mode. The acquisition frequency allows a frame rate of more than 200frames/s in VGA mode (640 x 480), in good lighting conditions.

The IEEE 1394 interface offers a real (exploitable) bandwidth of 20Mbytes per second from the camera to the external environment (host system), and 10 Mbytes/s in the other direction. A synoptic scheme of the hardware architecture is shown in figure 4.

5 Conclusions and Perspectives

This paper presented an architectural approach for a configurable processing element able to perform window-based operations. Element's conception is based on a computational model extracted from frequently employed image processing applications. Window-based operations being processing and data consuming, such kind of element presents a special interest for a smart camera platform. FPGA implementation allows to exploit massive parallelism, in order to reach real-time performances.

The proposed processing element architecture is implemented in the hardware platform described in section 4. The objective is to quantify the obtained performance and FPGA occupation for several values of n, in order to assess the validity and interest of this approach as part of a design methodology for embedded early-vision systems.

References

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach. Distributed embedded smart cameras for surveillance applications. *IEEE Computer*, 39(2):68–75, 2006.

[2] P. Chalimbaud and F. Berry. Embedded active vision system based on an fpga architecture. *EURASIP Journal on Embedded Systems*, 2007.

[3] F. Dias, P. Chalimbaud, F. Berry, J. Serot, and F. Marmoiton. Embedded early vision systems: implementation proposal and hardware architecture. In *Cognitive Systems with Interactive Sensors (COGIS)*, 2006.

[4] R. Kleihorst, B. Schueler, A. Danilin, and M. Heijligers. Smart camera mote with high performance vision system. In *International Workshop on Distributed Smart Cameras (DSC)*, 2006.

[5] H. Pourreza, M. Rahmati, and F. Behazin. Weighted multiple bit-plane matching, a simple and efficient matching criterion for electronic digital image stabilizer application. In *6th International Conference on Signal Processing*, volume 2, 2002.

[6] C. T.-Huitzil and M. A.-Estrada. Fpga-based configurable systolic architecture for window-based image processing. *EURASIP Journal on Applied Signal Processing*, 7:1024–1034, 2005.

[7] H. Yu and M. Leeser. Optimizing data intensive window-based image processing on reconfigurable hardware boards. In *IEEE Workshop on Signal Processing Systems Design and Implementation*, 2005.