Available online at www.sciencedirect.com
SCIENCE DIRECT®
Real-Time Imaging 11 (2005) 282–289
www.elsevier.com/locate/rti

Smart camera design for intensive embedded computing

Barthélémy Heyrman[a,*], Michel Paindavoine[a], Renaud Schmit[b], Laurent Letellier[b], Thierry Collette[b]

[a]LE2I, UMR CNRS 5158, Université de Bourgogne, Aile des Sciences de l'Ingenieur, BP 47870 - 21078 Dijon Cedex, France
[b]CEA/DRT-LIST/DTSI/SARC/LCEI, CEA-SACLAY, Bat 528, F-91191 Gif-sur-Yvette Cedex, France

Available online 24 June 2005

Abstract

Computer-assisted vision plays an important role in our society, in various fields such as personal and goods safety, industrial production, telecommunications, robotics, etc. However, technical developments are still rare and slowed down by various factors linked to sensor cost, lack of system flexibility, difficulty of rapidly developing complex and robust applications, and lack of interaction among these systems themselves, or with their environment. This paper describes our proposal for a smart camera with real-time video processing capabilities. A CMOS sensor, processor and, reconfigurable unit associated in the same chip will allow scalability, flexibility, and high performance.

1. Introduction

Due to the progress of technology, today it is possible to design a camera which integrates real-time image processing capabilities in order to capture not only the video data but pertinent information in the scene. Two main smart camera approaches have been proposed: (1) retina and (2) general-purpose vision chips. Retina consists of integrating some low-level image processing into the pixel. These smart sensors integrate focal plane processing elements made of tens of transistors doing image processing such as image quality improvement, image filtering, edge detection, movement estimation, etc. [1]. Indeed, image quality, for perception applications, is not only fidelity to the original optical image but the use of the full signal range in order to avoid saturation. While this is difficult to do with normal sensors, retina can easily satisfy these criteria. Classically, histogram equalization is one way to do this. In [2], the authors describe a smart sensor that makes histogram equalization during image acquisition for a cost of about 10 transistors. Another method, completely different, is proposed in [3]. It features a feedback loop that attempts to maintain a constant voltage across the photo-diode by giving it a current approximately equal to the photo-current. Early vision computations are edge detection, convolution, correlation, motion estimation, and velocity estimation. All these applications use physical phenomena in CMOS transistors such as diffusion and propagation. These applications are best adapted to highlight the advantages and the performance that we can reach with processing circuits in the focal plan. Movement information is provided in output of the sensor and can thus be used to set up an alarm (remote monitoring) or to generate a reaction (robot). Furthermore, interpretation of the movement (optical flow) calculated within the sensor will make it possible to control an autonomous vehicle: for example, without passing by an interpretation block. The benefit of compactness is obvious. In pattern recognition and object description, due to very complex shapes that cannot be embedded in hardware, programmable retina are needed. These kinds of retinas can be considered as SIMD array processors. In [4], the description of a smart sensor performing binary operations is given. Operations are boolean (AND, OR, XOR) as well as morphological

*Corresponding author. Tel.: +33 380393827.
E-mail address: barthelemy.heyrman@u-bourgogne.fr (B. Heyrman).

1077-2014/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2005.04.006

ones. Another application is fingerprint sensing and identifying [5]. In this circuit, sensing elements are stacked above the identifier. The result is that each pixel has is own identifier. The circuit has a performance time of 102 ms for sensing and identification. In the work presented previously, the vision chips were designed to perform a specific process or very simple operations.

The second smart sensors architecture solution couples sensors and general-purpose processors. This allows complex applications and flexibility. Often, a vision algorithm needs more than one process; retina can help but are not sufficient. All the works described below implement general-purpose processors. In [6], the authors describe a vision chip where each pixel contains a processing element composed of a bit-serial ALU, a 24 bit random access memory and an 8 bit memory mapped I/O. This circuit performs edge detection in 5.6 μs and smoothing in 7.7 μs. The SCAMP architecture [7] is a mesh array of analogue processing elements (APE). A 21×21 SCAMP vision chip has been fabricated and performs smoothing in 5.6 μs and edge detection in 11.6 μs. The chip presented in [8] is especially designed for block-wise transformation (DCT) or spatial convolution. It performs operations using an 8×8 kernel. A 128×64 pixel sensor has been designed in 0.6 μm technology. Another chip based on cellular neural networks (CNN), called ACE16k [9], uses a 128×128 cell array. Each processing element (PE) is a Mixed-Signal SIMD-CNN that is able to compute 3×3 convolutions, boolean and arithmetic operations, and CNN-like evolutions.

We see in this section that research into vision-chips is active. However, the presented solutions (retina and general-purpose processors) seem to have difficulties in fulfilling all the vision applications. The retinas offer the advantage of compactness and are very efficient for early-vision process, but due to limited space, functionality is very poor and often limited. Vision chips based on general-purpose processors enhance possibilities due to programming features but their performance is inferior in terms of size and speed. Considering the constant evolution of CMOS technologies, integrating a full digital processing array into a sensor chip is now plausible.

In this paper we present our proposal for a smart camera. We choose to integrate in the same chip a sensor with high parallel outputs and an SIMD array of processors. Our processor model is specifically dedicated for vision application and is optimized for some criteria like silicium area and speed-processing. In Section 2 we describe the architecture of our smart camera, and in Section 3, we present the modeling and validation of this architecture.

2. Camera design

As we have seen in the previous section, the two approaches described are not satisfactory. Indeed, retinas, even if they offer processing levels very close to sensors, cannot deal with complex applications. They are well suited for early vision processing and can be useful in speeding up processes. General-purpose approaches, if they allow more complex algorithms, do not have the same performance as retina.

A second remark concerns the readout capabilities of vision chips. Although they perform their computation in an SIMD scheme, they often send pre-processed data sequentialy. This is an important bottleneck in the datapath and reduce considerably the available bandwidth.

The third remark we can make is that there are few pertinent data in an image. In image computing, most applications work on objects of interest, e.g. Data-Matrix, land-mark detection, people tracking, finger print recognition. Consequently why process the remainder of the image?

In order to answer these questions, we choose to use a sensor with Region of Interest (ROI) readout and parallel outputs capabilities. It is linked to RAM or an array of processors that are able to store/compute provided data in order to provide an efficient way of acquiring and computing ROIs. We give the scheme of our architecture in Fig. 1 and the following sub-sections describe more accurately the different parts.

2.1. Sensor with multiple digital video outputs

The sensor is based on CMOS technology which allows random access to the pixels. However, a more important feature is being able to access an image sub-region in a single step. The same concept (ROI) is used in [10]. The authors present a sensor that can provide regions in outputs but also it can do tracking. The size and position of ROI can only be changed by steps of 32 pixels in order to save chip area. However, a more

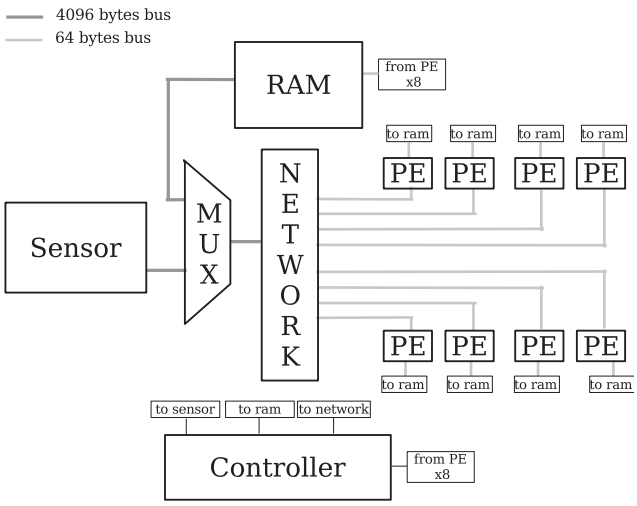

Fig. 1. Modeled architecture scheme.

accurate step can be useful in motion-based algorithms. Data are read 4×4, and although it is better than 1×1, this does not offer a very high-speed bandwidth. In order to answer this, the sensor is designed to provide 4096 pixels in parallel. Indeed, future CMOS technologies (90 nm and less) will not only reduce the size of transistors but also the size of links in metal layers. In our case, we want to integrate all of our architecture into the same chip. Submicronics technologies can help us—it is possible to output many more links inside the chip than outside it. For example, in 90 nm process, the size of a link in metal 2 is 0.30 μm and the pitch is 0.28 μm. That means that our 4096 bytes require about 20 mm of silicium to be output in metal 2 layer, i.e. a square of 25 mm^2. The readout circuit allows only square windows up to 64×64 pixels in one clock cycle to be sent to the RAM or to the network. Another advantage, outputting in parallel, is the reduced operation frequency of the AD converters, and thus they can be simpler.

2.2. Choice of network

The choice of our network is an important part of our system. Indeed, if we are not able to efficiently provide the data to processing elements, or if the link between sensor and PE array is a bottleneck, the system will suffer from an important loss of performance. Cross-bars are efficient systems for providing data, but their cost in area increases with n^2 when the number of inputs is high. So, we decide to use a partial cross-bar. Its rule is to dispatch data coming from the memory or the sensor to the processor array. In the 64×64 window it selectively chooses a smaller window of up to 8×8 pixels from anywhere inside the larger window. It can do this operation for the 8 PEs (processing element) at the same time. This means that 64×8 pixels can be sent within a clock cycle.

2.3. RAM design

In this system, the choice of dual-port RAM seems to be an interesting solution. Indeed, such memory can read and write at the same time. The memory capacity is 128×64 bytes. One port linked to the sensor or the network via the multiplexer is 4096 bytes large. This allows the memory to store a window in one clock cycle. Its operation is classical: a clock, an address word, and read/write signals. The second port is quite different and is 8×64 bytes large. It is linked to each PE. It has a clock, read/write signals and a command word. A scheme of the RAM inputs and outputs is given Fig. 2.

2.4. PE description

These PE are designed to execute as efficiently as possible some basic operations often found in vision

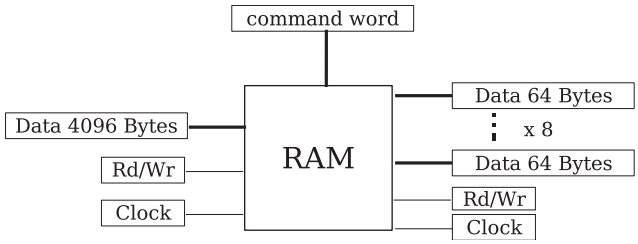

Fig. 2. Scheme of the RAM.

applications. Indeed, vision applications are focused on one or more very small parts of the image. Examples of application are the JPEG compression that uses 8×8 windows or land-mark detection. That is why we have decided to design a processor that can achieve operations on windows. We choose a 8×8 maximum window size that can be processed by a PE but we can envision grouping the PEs to process larger areas. Implemented operators are arithmetic and logic, mask filtering, convolution, and correlation. The operators can be used variously. Arithmetic operation can be done between two different windows or between a window and a constant. Each 8-bit processing core in Fig. 3 corresponds to the scheme presented in Fig. 4. These cores are divided into three parts. The first part is an 8-bit Arithmetic and Logic Unit (ALU). It works on integer values for this first implementation. The second part is a shift register that performs a power of 2 division. For the third part, we chose to use MAC structure in the filtering unit in order to process data efficiently. We have also added three registers, called FRC1, FCR2, and FRC3 in Fig. 4, in order to store coefficients of different filters. These registers can only be used by the Mask filtering unit. It also contains three shared byte registers (R1, R2, and R3 in Fig. 4) in order to store data between two operations and 8 internal 8 bits registers to store data used in computation. Despite having three different units in the core, these units cannot currently operate at the same time. For example, in addition, a mask filter operation, and a shift is not possible. The global PE also contains a 256-byte register for table operations (like histograms), and a code-cache in order to store the applications. Available operators are given in the Table 1.

Applications are written in pseudo-assembler code and are all coded like this:

- A key word: *avr* for averaging, *add* for addition, etc.
- A string giving the correct 64×64 window to be outputted by the sensor.
- A string indicating the memory bank to be accessed.
- 4 numbers giving the configuration of the network: XY position and XY size of the window to be outputted.

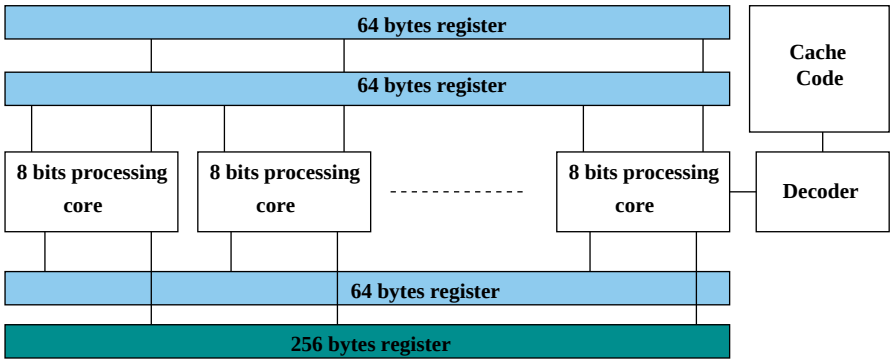

Fig. 3. Simple scheme of a PE.

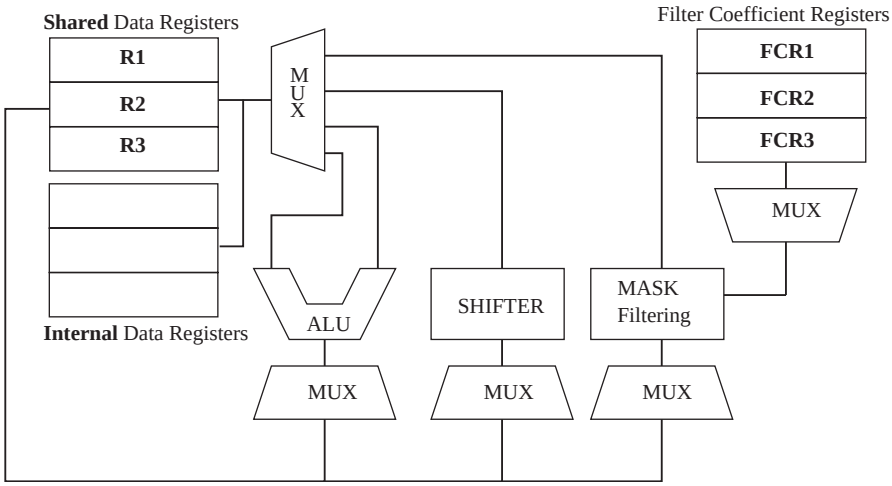

Fig. 4. Detailed description of a 8 bits-processing core.

Table 1
Instructions availables in the processors

Instruction	Description	Instruction	Description
adr	Addition registers function	add	Addition function
mul	Multiplication function	divr	Divison function
abs	Absolute value function	sqr	Square value function
impd	Import data function	ld	Load constant in register
cp	Copy data function	cmp	Compare function
max	Maximum function	or	Or function
and	And function	ml	Load data from memory
sum	Sum function	dec	Decrease function
inc	Increase function	bne	Branch if not equal
bm	Branch is greater	sub	Substraction function
sto	Store in memory function	avr	Average function
mas	Filtering function	thr	Threshold function
not	Logical not function	end	Stop function

For example

add r0 m0 0 0 8 8 // addition.
hst // histogram of the preceding result.
thr 45 // threshold according to the value given in argument

will do the addition of the upper-left window of the sensor image and a upper-left window of an image stored in memory bank 0. Some instructions like histogram computation or image thresholding are directly done in the image stored in memory and do not require additional parameters or data. A very simple scheme of a PE is given in Fig. 3.

2.5. Controller task

The controller's role is to manage all the other parts of the system. It generates command signals depending on requests coming from the PEs. It drives image acquisition and storage in memory, sends start commands to the PEs, and manages all the access to the memory. It can be compared to an interrupt controller. While waiting for requests or events from all the other parts of the system, it does not process anything but it is programmed, depending on the application, to acquire and store one or more image. Its exact operation will be described in detail in Section 3.

3. Architecture modelization and validation

In order to study our concept we decided to model it in System C [11]. Simulation times are faster using this C++ library than in VHDL. Another advantage is the high-level description possibility. So we can make a full software functional model as a first step and as a second step we will refine it using hardware software co-simulation. In order to validate our architecture, we tested it on a motion detection application. We present hereafter first the application, second some parameters used in the model, third the active parts of the architecture during the runtime, and fourth the results.

3.1. Application description

This application concerns simple motion detection. In order to do this, we first make an average of 4 images in order to reduce noise and after we substract the average from a reference image. The result is thresholded and some morphological operations (erosion, dilation, erosion again to eliminate residual noise) are done. We count the number of moving objects, if this number is different from the reference, the reference image is updated. This application is illustrated in Fig. 5.

3.2. Preliminary

We present now several global parameters used in our model. They are used to configure the model operation. All are in binary representation

- **Sens_PosX and Sens_PosY** correspond to the upper left corner of the window outputted by the sensor.
- **Net_WidthX and Net_WidthY** give the size of the window transmitted by the network.
- **Net_PosX and Net_PosY** code the position of the outputted network window in the sensor window.
- **Mem** gives the memory bank address.

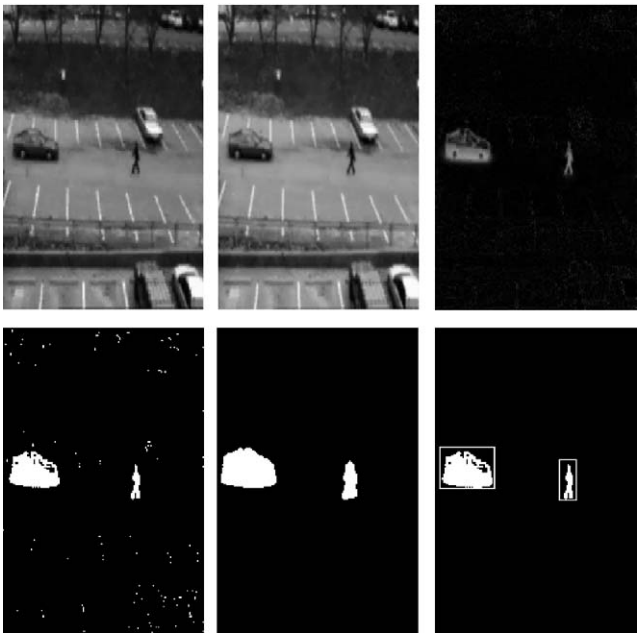

Fig. 5. Motion detection example (from left to right: input image, average of 4 images, substraction, threshold, morphological operations, result: labeling).

3.3. Model operation during runtime

After a reset the controller sends an acquisition signal to the sensor. It starts image acquisition according to the programmed exposure time and returns an "end acquisition" signal. This is the first step as described Fig. 6a.[1] When the controller receives the "end of acquisition" information, it starts, in the case of our application, a memorization sequence, i.e. it sends a ***Sens_PosX*** + ***Sens_PosY*** word to the sensor. The LSB bits are the X-coordinates of the window outputted and the MSB bits are the Y-coordinates as shown in Fig. 7a. After two clock cycles[2] valid data are available in the output of the sensor and the controller sends a ***Mem*** bits address word to the memory to store the data. At this time the multiplexer is configured to transmit data between the sensor and the memory. This is the second step shown in Fig. 6b. When acquisition is done, the controller sends a start command to the PEs as shown Fig. 6c. The PEs start their code execution and depending on the instruction, generate a request. A request is a 32-bit word

- The LSB bit indicates that requested data should come from the sensor (if 1) or memory (if 0).
- The ***Net_WidthX*** + ***Net_WidthY*** next bits gives the size of the window for the network.

[1] All gray parts in Fig. 6 illustrate active parts in the system.

[2] This value has been chosen arbitrarily in order to simulate delay in data transmission.

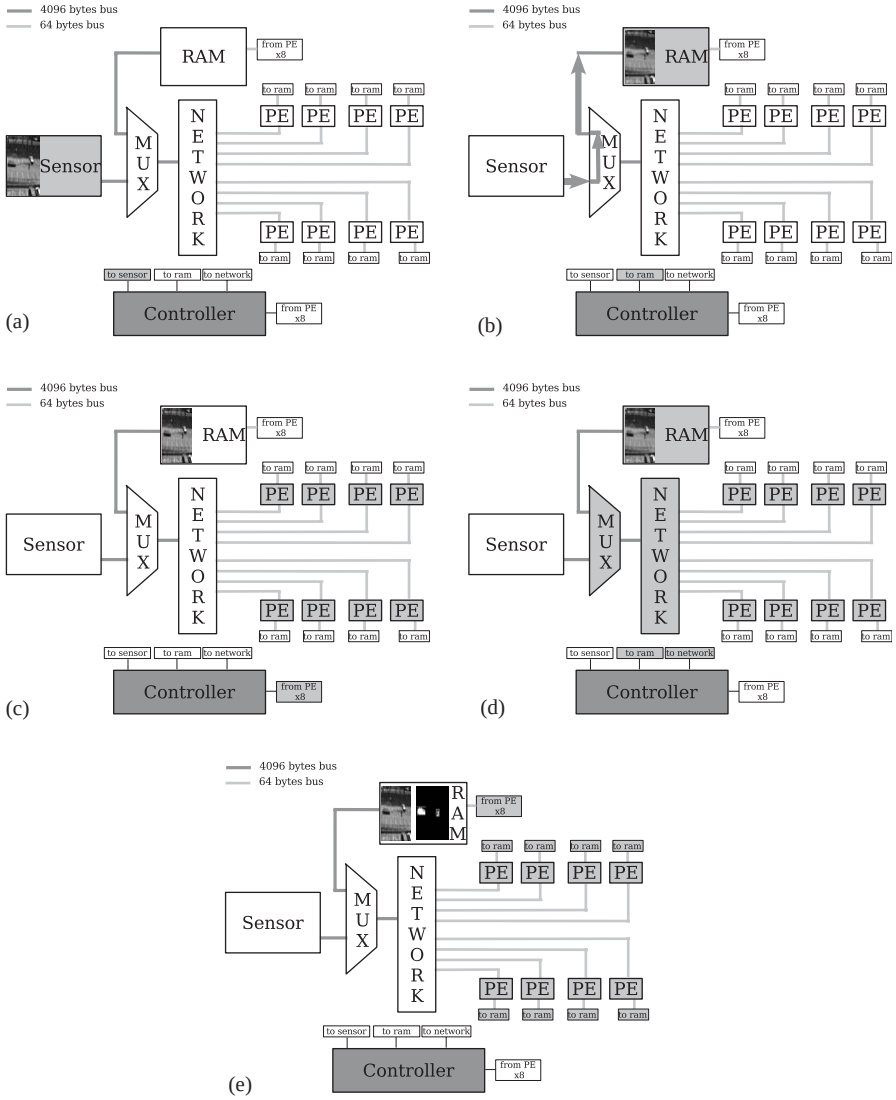

Fig. 6. System operation sequence: (a) Step 1: image acquisition, (b) Step 2: image storage, (c) Step 3: PE ask for data, (d) Step 4: data are sent to PEs and (e) Step 5: result.

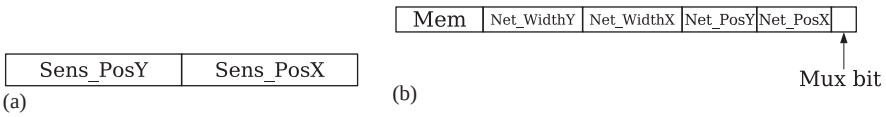

Fig. 7. Sensor command and request word structure: (a) sensor command and (b) request structure.

- The ***Net_PosX*** + ***Net_PosY*** next bits gives the coordinate of the top left corner of the window in the network.
- The ***Mem*** next bits correspond to the memory address *bank*.

If the PEs ask for data from the sensor the controller sends a command to the sensor to output the correct window. It puts the multiplexer in the right configuration and sends a configuration word to the network. If memory data are requested, an address is sent to the RAM instead of the sensor and the data path is configured to transmit data from the RAM to the PEs. This case of operation is shown in Fig. 6d. As long as the program does not meet an *end* instruction, the functioning states are Figs. 6c and d: request and answer. When PEs have finished their process (in the example, averaging of 4 images, substraction, thresholding, and morphological operations), they store

their results directly to the memory (see Fig. 6e). They send a command word coded like this:

- the ***Net_WidthX*** + ***Net_WidthY*** bits for the window size.
- ***Net_PosX*** + ***Net_PosY*** bits for the window position in the memory bank.
- ***Mem*** bits for the bank.

Finally, PEs send an "end of process" signal to the controller. When all PEs are inactive a new image can be acquired and the process sequence restarted. Our model is configured with the following parameters.

- **Sens_PosX** = **Sens_PosY** = 9 bits.
- **Net_WidthX** = **Net_WidthY** = 4 bits.
- **Net_PosX** = **Net_PosY** = 6 bits.
- **Mem** = 5 bits.

3.4. Results

The diagram shown in Fig. 8 illustrates the operation of our model. It shows, in the case of motion detection application, the two important sequences of our model. The first one is the image acquisition. In this configuration, we see that the active signals are *adress[5:0]*, *addr_ecr_mem[5:0]*, ... showing acquisition of image and storage to the RAM. The second part is the image processing: here all the signals are active especially the 8 *commande[19:0]* signals (command to the network) and *demande* signals (corresponding to a resquest from a PE to the controller).

The first result we obtained is the acceleration of the data storage in memory. It is done with steps of 4096 pixels. This means that if we use a 10 ns clock for the system and $1\,K \times 1\,K$ sensors, 256 clock cycles are needed to store a full image. The theoretical maximum bandwidth between the sensor and the RAM is $2.56\,\mu s \times 1\,K \times 1\,K = 409.6$ GBytes. It is interesting to note performance obtained with this architecture. An operator needs 3 clock cycles in order to output data processed in the worst case (data need to be received and stored for all the instructions): one to receive data, one to process them, and one to send them to memory. For example, a dataflow application with 5 different tasks requires 15 cycles to process 8×64 pixels. So a $1\,K \times 1\,K$ image will be processed in 30 720 cycles:

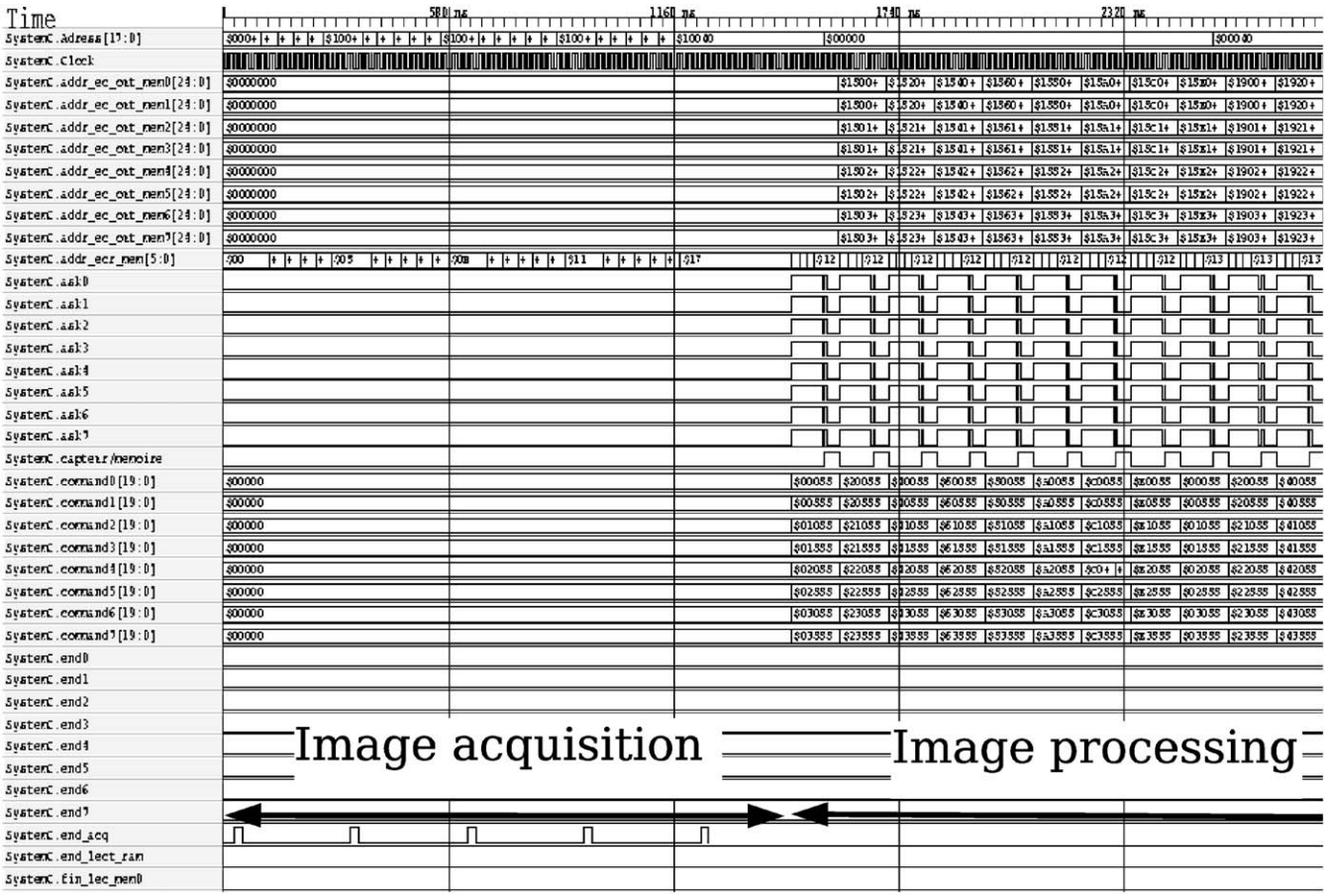

Fig. 8. Functioning diagram of the system.

$(1\,K \times 1\,K)/(8 \times 64) \times 15$ cycles. If we choose a 10 ns clock, it will take around 0.3 ms to finish the treatment. In this case more than 3000 images could be processed in a second.

4. Conclusion and perspectives

We have described a new architecture for a smart camera offering very high-speed processing capabilities. This is done by designing dedicated parts such as sensors with massively parallel outputs and ROI readout circuits. The processor array takes care of vision applications and has specific operators such as mask filtering, convolution, etc. A fully functional software model has been created in SystemC and tested using a motion detection application.

Future works will be other test applications implementation on the architecture (JPEG compression, motion estimation, etc.). A second part could be the insertion in the network of one or more floating point units in order to target applications that require more accuracy. The reconfigurable logic in the PE that can help us to design some specific processes even after a chip design are not yet integrated. We plan to make a prototype with reduced parallel output (16×16 parallel outputs). A VHDL translation of the SystemC architecture model is in progress. The processor cores need some work in order to operate well. This VHDL code will be implemented in a 0.35 μm CMOS process using Mentor Graphics software.

References

[1] Moini A. Vision chips or seeing silicon. Technical report, The Centre for High Performance Integrated Technologies and Systems, The University of Adelaide, March 1997.
[2] Ni Y, Devos F, Boujrad M, Guan JH. Histogram-equalization-based adaptative image sensor for real-time vision. IEEE Journal of Solid-State Circuits 1997:32(7).
[3] Delbrück T, Mead C. Analog VLSI phototransduction by continuous-time, adaptive, logarithmic photoreceptor circuits. 1995; 139–61.
[4] Paillet F, Mercier D, Bernard T. Making the most of 15 K lambda2 silicon area for a digital retina, In: Spie P, editor. Conference of advanced focal plane arrays and electronic cameras, vol. 3410, 1998.
[5] Shigematsu S, Morimura H, Tanabe Y, Adachi T, Machida K. A single-chip fingerprint sensor and identifier. IEEE Journal of Solid-State Circuits 1999; 34(12):1852–9.
[6] Ishikawa M, Ogawa K, Komuro T, Ishii I. A CMOS vision chip with SIMD processing element array for 1 ms image processing. In: Proceedings of the conference ISSCC'99, No. TP2.2, 1999.
[7] Dudek P, Hicks PJ. A general-purpose CMOS vision chip with a processor-perpixel SIMD array. In: Proceedings of ESSIRC2001, 2001.
[8] Graupner A, Schreiter J, Getzlaff S, Schüffny R. CMOS image sensor with mixed-signal processor array. IEEE Journal of Solid-State Circuits 2003;38(6):948–57.
[9] Rodrigues-Vazquez A, Linan-Cembrano G, Carranza L, Roca-Moreno E, Carmona-Galan R, Jimenez-Garrido F, Domiguez-Castro R, Espejo Meana S. Ace16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSOCS. IEEE Transactions on Circuits and Systems 2004;51(5):851–63.
[10] Schrey O, Huppertz J, Filimonovic G, Bussmann A, Brockherde W. Hosticka B. A 1 K × 1 K high dynamic range CMOS image sensor with on-chip programmable region of interest readout. In: Proceedings of ESSCIRC2001, 2001.
[11] System community web site. http://www.systemc.org