FPGA Lab Sessions in a General-Purpose Image Processing Course

Philippe Guermeur[1], Petr Dokladal[2], Eva Dokladalova[3], Antoine Manzanera[1]

1) ENSTA, LEI, 32, Boulevard Victor, 75015 Paris, France, philippe.guermeur@ensta.fr, antoine.manzanera@ensta.fr

2) ENSMP, CMM, 35, St. Honoré, 77305 Fontainebleau, France, petr.dokladal@ensmp.fr

3) ESIEE, A2SI, 2, boulevard Blaise Pascal, 93162 Noisy le Grand, France, e.dokladalova@esiee.fr

Abstract This paper presents an advanced course for CS master program students who have a solid background in image processing and partial (or none since the group is heterogeneous) knowledge in the logic design.

This course presents advanced notions like data synchronism/availability, parallelization, access to the pixel neigborhood, rethinking the algorithm to make it more hardware-efficient (sharing ressources, using parallel data flow paths).

We illustrate these notions on a contour detector using the zero crossing of the Laplacian.

1 Introduction

The image processing represents a large field with numerous applications (object and scene recognition, data mining, compression, image synthesis, and many others). These applications are implemented by very different algorithms coming themselves from various mathematical frameworks.

Even if the image processing algorithms are often of a reasonable complexity, the considerable amounts of data to process, real-time constraints, data structures and their dependency make the requirements on the hardware very specific (a good analysis can be found in [5]).

The widely ignored fact is that the major concern is not the data processing but rather feeding the processor with the data. Many algorithms need random (!) access to data in several images. The image size makes difficult to store several images in the cache of the processor and one generally cannot run real-time image processing applications on small processors like ARM or NIOS, without coupling them with some specialized hardware block [1]. Digital signal processors (DSPs) are adapted only to a limited class of algorithms (linear processing) [5] . In the context of embedded systems, one often needs specialized hardware. The design of dedicated image processing hardware (HW) requires skills from both domains: the logic design and the image processing.

2 Educational objectives

The FPGA labs presented in this paper take place within an Image Processing and Artificial Vision course of 12 sessions given at ENSTA (Paris). The general philosophy of engineering education at ENSTA is to train generalist engineers with a broad knowledge, and able, at the same time, of a deep involvement in every technical aspect of their current project.

In that spirit, the Image Processing course has a general purpose. It is at the crossroads of various elective curricula : Multimedia, Robotics, Embedded systems, Information systems. It aims at providing the students with a solid background in linear and non linear image processing (filtering, segmentation, colour image processing, scale-space, partial derivative equations models, motion analysis, 3d reconstruction, shape analysis).

The practical sessions (computer labs) associated with this course (5 sessions over 12) have a three-level ambition:

• **experimental:** the student can, with a minimal programming investment, apply the operators seen in the theoretical sessions on his (her) own data. At the same time, it allows to introduce the notions of systems specifications and its integration.

• **algorithmical:** the student must understand the transition between the mathematical and the numerical models ; (s)he is expected to propose a data structure, program the algorithm and think to the consequences in terms of complexity.

• **real-time:** the student must be aware of the adequacy of the data structure and the algorithm to the hardware it runs on, in terms of real-time and compactness of the system.

This multi-level objective implies multi-platform labs: Matlab, C, SIMD-within-register (SSE-2), and finally FPGA boards are successively used in the practical sessions.

The purpose of the FPGA labs is to provide an advanced insight into the issues of real-time, embedded image processing. For the reasons given in

Section 1, we think that this knowledge is important for any engineer involved in a video application. But it is a challenging purpose because of the various curricula of the students: if all of them have a basic background in digital electronics, only a few have any practical experience on FPGAs.

At this point, we can consider two alternate solutions:

- either using a higher level framework (e.g. Simulink or Celoxica) to generate the logic synthesis. The advantage of this solution is to be more rapidly accessible to all the students, but the risk is to hide the difficulties and thus limit the reflections of the student on the embedded video system constraints.
- or using a hardware description language (e.g. VHDL or Verilog) to be closer to the logic blocks. The advantage is that it is easier to make the student work on a given specific aspect of embedded image processing system, but this implies a high degree of preparation of the lab session, in order to make the practical work accessible to a student with a small knowledge of FPGAs.

We have chosen the second solution, and then the problem we are going to develop in the following is: how can we propose a deep involvement in the real-time embedded image processing system, including real experimentations with a camera, while making, as much as possible, abstraction of the hardware description language and of the platform specific constraints ?

The real-time issues we typically wish to be considered by the student are the following ones:

• **memory management:** what is the amount of memory that needs to be available by the algorithm at the same time?

• **time management:** how to manage real-time considering the clock frequency, the computation time, and the memory delays?

• **processing model:** what is the available instructions set, and what are the operands (fixed vs floating point)?

• **computation paradigm:** data flow computation, data parallelism, pipe-line.

• **data structure:** how many different data representations are there and what granularity is more adapted?

Our choice for a suitable subject has been made on the following arguments:

- fast-to-implement and interesting application: the algorithm should not be too low-level, because the student needs to understand its interest as an embedded module. In that sense, an application leading to a massive data reduction is preferable to image filtering or enhancement. At the same time, the algorithm must be implementable in the allotted time.
- containing blocks with various complexity to implement: we can take advantage of the heterogeneity in the group of students to make them work in pairs or groups of three on different aspects of the project.
- algorithm that does not immediately fit HW implementation needs, but which might be slightly modified in order to share ressources: an additional optimization margin must be available in order to increase the interest of the labs and allow a greater variability between the different groups.
- containing reusable generic blocks: students can see in that case the incremental and hierarchical capabilities of sequential logic.

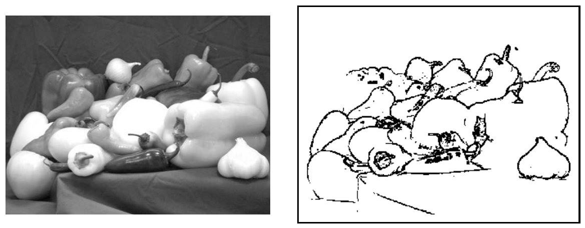

Figure 1: Contour detection by zero-crossing of the Laplacian.

Taking into account all these features we have chosen a contour detection algorithm running in real time. This application represents an sufficient data reduction to justify its implementation on dedicated HW. It is complex enough to lend itself to a multiple level data flow description, containing blocks of different complexity. We have chosen to use the basic operators of Mathematical Morphology, which leads naturally to a description using generic blocks, which can be reused, or even shared.

The input video is furnished by a B&W camera. The results can immediately be seen on a VGA display. See an example of the input image and the contours detected on Fig 1.

3 Algorithm description

The contours are detected by detecting the change of sign of the Laplacian [3] (or [8] for more details). The two elementary operators used are the dilation δ and erosion ε. Let $f : \mathbf{D} \rightarrow \mathbb{Z}$ be the input image defined on some domain $\mathbf{D} \subset \mathbb{Z}^2$. The erosion εf and dilation δf of f in some pixel $x \in \mathbf{D}$ is :

$$\varepsilon f(x) = \min_{x_i \in B(x)} \{f(x_i)\}$$

$$\delta f(x) = \max_{x_i \in B(x)} \{f(x_i)\}$$

given some so-called *structuring element* $B \subset D$, $B(x)$ denotes the translation of B by x.

The so-called *external* and *internal* gradients are $g_{\text{ext}}(f) = \delta f - f$ and $g_{\text{int}}(f) = f - \varepsilon f$. The morphological gradient g and morphological Laplacian $\mathcal{L}$ are then $g = g_{\text{ext}} + g_{\text{int}}$ and $\mathcal{L} = g_{\text{ext}} - g_{\text{int}}$.

The change of sign in $\mathcal{L}$ (zero crossing) is obtained by $\mathcal{L}^{\pm} = \delta\mathcal{L}^{+} \cap \delta\mathcal{L}^{-}$ with $\mathcal{L}^{+}(x) = \{x \mid \mathcal{L}(x) > 0\}$ and $\mathcal{L}^{-}(x) = \{x|\mathcal{L}(x) \leq 0\}$. The contours c in the image are obtained $c = \{x \mid x \in \mathcal{L}^{\pm}$ and $g(x) > \text{Th}\}$ where Th is a contrast threshold filtering out weak contours (see e.g. the "peppers" image and the detected contours at Fig. 1).

In this application, the structuring element B will be the eight neighborhood of some pixel $x(i, j)$: $B(x(i, j)) = \{x(i\pm1, j\pm1)\}$. We will enumerate the current pixel and its eight neighbors by subscript indices $x_{11}, x_{12}, \ldots, x_{33}$.

The dilation and erosion are dual operations, operating locally on the neighborhood of a point. These operations serve in this application twice: first, to compute the gradient in the image, and second, to compute the laplacian $\mathcal{L}$. We add a threshold to control the level of necessary contrast for an edge to be accepted by the detector.

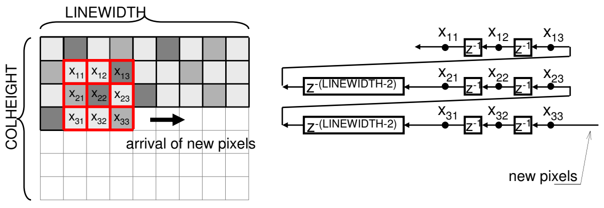

Figure 2: Neighborhood extraction.

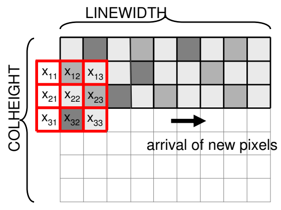

Figure 3: Missing values when processing the image border pixels.

Implementing dilation/erosion requires the extraction of the neighborhood of the current pixel x_{22}, see Fig. 2, and computation of max/min on these values. For processing the image borders, one needs to substitute the value of the outlying pixels (see Figure 3) by some value that will not affect the result (i.e. $-\infty$ for dilations and $+\infty$ for the erosions). The line and column counters detect the border of the image. Substituting the missing values in the set of the neigbors values is the role of the edge handling block.

4 HW implementation costs

From many years, morphological operators are implemented on different HW platforms, we can cite [9] for one of the first dedicated architectures, [2], [7] as examples of more recent custom computing or [6] for reconfigurable approach.

In this implementation, no memory is used for storing the entire image. The pixels are processed on the fly during the video scanning of the input image. See Fig. 2 left, the central pixel x_{22} is processed and output once its complete neigborhood is known, i.e. as soon as its latest pixel (x_{33}) is read. Therefore, the one-line latency introduced by the algorithm is due to the fact that all the necessary data are available only once the following line is read.

Hence we are in the context of SISO (Stream Input Stream Output) image processing applications. If some algorithm allows the SISO implementation, it can be implemented with a very low memory cost and a very low latency of the processing.

Surface occupation Various blocks have various surface occupation cost. The neigborhood extraction block needs to store two complete rows of the image, see Fig. 2 right. These rows need to be stored in a delay line the size of which is exactly the width of the image (LINEWIDTH). Two registers z^{-1} are used separately and the rest in a block of LINEWIDTH-2 units (denoted by $z^{-(\text{LINEWIDTH-2})}$).

Hence, one dilation/erosion block needs to store 2 rows of the image. Its processing latency is 1 image row. The subtraction of the dilated image and the original also needs to compensate this delay: one image row of storage. The implementations costs resume to Table 1.

The by far most costly block is the neighborhood extraction. Its cost depends on the image size.

It is important to show to the students the very basic fact that this allows outputting the output image frame even before the input image frame is entirely read. On the other hand, a simple subtraction of image and its erosion requires adding a

Table 1: Implementation costs.

two 8-bit δ/ε blocks : - four 8-bit FIFOs (size: 1 image row)
two 1-bit δ/ε blocks : - four 1-bit FIFOs (size: 1 image row)
two delay compensations : - one 8-bit FIFOs (size: 1 image row) - one 1-bit delay line (size: 1 image row)

block compensating the latency of the erosion.

Optimizing the dataflow Dilation and erosion are two dual, set-wise (in this context) operations. We have $\mathcal{L}^+ = (\mathcal{L}^-)^c$, where X^c being the complement of X. The duality comes from the identity $\delta X^c \equiv (\varepsilon X)^c$. However, the equation

$$\delta X \cap \delta X^c = \delta X \cap (\varepsilon X)^c \tag{1}$$

doesn't have the same surface occupation on both sides. If dilation and erosion run on the same data (as in the first stage of the Laplacian) they can share the most costly neigborhood extraction block, see Fig. 4.

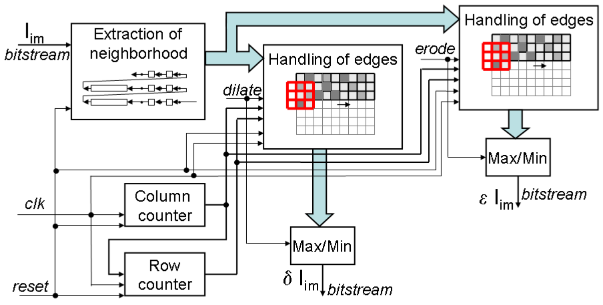

Figure 4: Dilation and erosion share the neighborhood extraction block.

Modifying the algorithm according to Eq. 1 allows a considerable reduction of silicon occupation, see Table 2.

Table 2: Optimized implementation costs.

two 8-bit δ/ε blocks : - two 8-bit FIFOs (size: 1 image row)
two 1-bit δ/ε blocks : - two 1-bit FIFOs (size: 1 image row)
one delay compensations : - one 1-bit FIFOs (size: 1 image row)

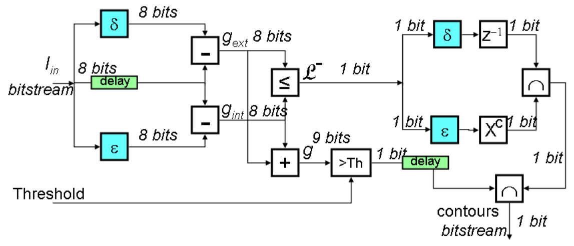

Figure 5: Algorithm modified using Eq. (1). X^c denotes complementation.

5 The educational platform

5.1 The hardware environment

The specifications of the hardware platform are set up to provide convenient resources for a great variety of student projects. This versatility is desired to develop many real time image projects without multiplying the material equipment and then to reduce the complexity and the cost. This should enable the students to get more familiar with a given complex environment and get a better understanding of their FPGA hardware platform. As a consequence, students are expected to go deeper in conceiving their embedded image processing system.

These educational purposes lead us to define the following system requirements specification:

• **inputs-outputs**: conveniently, the platform has to contain hardware for enabling video acquisition and a VGA output is required for a rapid testing and debugging of implementations.

• **local memory**: the typical applications to be developed concern basically low-level image processing. Most of these applications (convolution, recursive filtering, wavelet transform, ...) can be performed on a stream of data, using a neighborhood processing. This neighborhood could be implemented using FIFOs as it was done in many DSP components in the 1990's (Thomson IMSA110, Texas C80...). In 1998 we have proposed a solution to implement these resources using dual-port static RAMs [4]. The advantages of the dual-port solution was to propose reusable hardware when no neighbourhood is needed but other kind of resources are required (LIFOs, data passing between tasks...). Nowadays, the architecture of the XILINX components has evolved and the Spartan series, for instance, integrate columns of dual-port RAMs, which make them much more appropriate for image processing.

• **global memory**: usually, low level image processing is not an end in itself. It transforms a video sequence in another video sequence, but the produced data are too bulky and not very practical to handle. A data transformation is necessary to get an intermediate knowledge representation (Freeman chains, quad-tree, image moments...) which can be better digested in the image understanding tasks. While this intermediate level image processing usually requires peculiar image scan (such as content-based scan or Peano scan...) we decide to acquire a platform equipped with a memory space big enough to store a few images.

In 2005, only a few commercial boards offering the previous hardware resources were available. We selected a platform built by XESS Corporation composed of the XSA3S1000 board [10] (the processing unit) combined with the XST-3 board providing analog and digital I/O capabilities (video input, USB, Ethernet...). This platform (see Fig. 6) contains a 1,000,000-gate Spartan3 component, 32 MByte SDRAM and a VGA output.

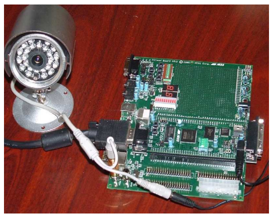

Figure 6: The development board.

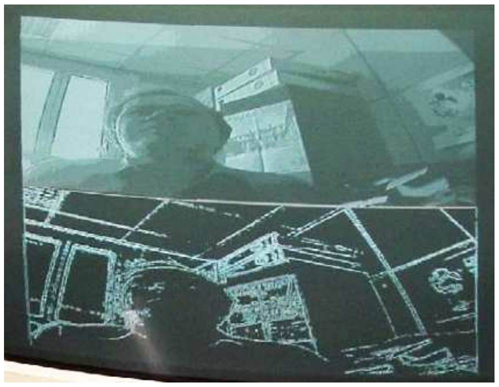

Figure 7: The results on a VGA monitor: the camera stream (above), the contours (below).

5.2 The integration environment

The software environment must allow the students to get a rapid familiarization with the Design Implementation tools, and the associated HDL design flow. So, the main labs concern is to provide the students with a simple interface keeping them out of the menial task of managing the various input-output of the board (board configuration, communication protocols, multi-port SDRAM interface, image acquisition and VGA display...). This enables the student to concentrate on the more satisfying task of conceiving the core of the image processing unit (black-box) in Fig 8.

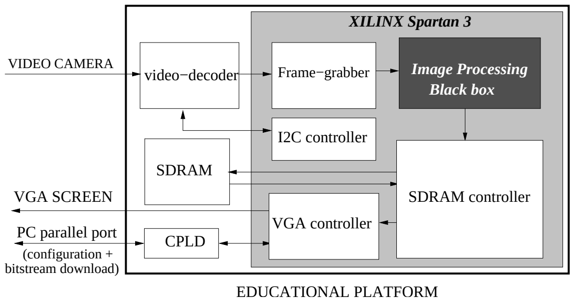

Figure 8: Schematic representation of the various data managements.

The side effect is that we had to cope with the difficult task of designing and implementing such a simple interface. This lead us to modify or implement and test various component cores (multi-port SDRAM controller, framegrabber, VGA core...), to develop some drivers and to model some components (video bus model, SDRAM...) and test the whole simulation environment. This environment reveals to be a great help for testing or debugging students implementations. While simulation can illustrate them the functioning of essential element of the board, it is supposed to help the most motivated students to go deeper in the understanding of the platform. In the context of SISO, images could be displayed as soon as they are processed Fig 9(a), but such a process leads to ugly displays when using interlaced cameras:

- image flicker appears when the odd fields are displayed below the even fields (or vice versa);
- milled contours appear when the fields are interlaced.

A VGA interface composed of a RAMDAC should have avoided this drawbacks. Here, there is no RAMDAC available and the solution is to store the

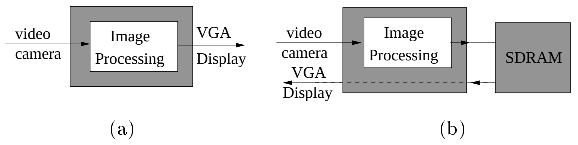

Figure 9: Pixels are processed on the fly: a) no memory and no RAMDAC is available (poor quality rendering). b) intermediate storage allows to eliminate the visual defects.

image data in the SDRAM, as soon as they are processed Fig 9(b). A very simple protocol has been built to make this step transparent for students. In the context of SISO, the processing unit they have to implement is the content of a black box whose inputs are composed of the data bus (pixels) and two control signals (Fig 10) and the outputs are the processed data and the two delayed control signals.

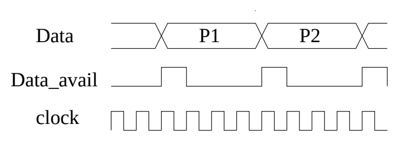

Figure 10: Protocol defined at the input of the image processing unit.

6 Organization of the course

The course consists of an introductory 3h lecture, followed by two 3h30 lab sessions.

Lecture: Students are presented the algorithm, get familiar with the data flow, estimate the surface occupation cost, and modify the algorithm according to the identity Eq. (1) in order to reduce the implementation costs.

The Laplacian is computed in two stages. These two stages are similar, since they use same operators : morphological dilation and erosion, addition, subtraction and/or thresholding.

The first stage of the Laplacian operates on greyscale data whereas the second one on binary data. This allows to stress out various important aspects of HW design and makes the students think about the correct data widths of various operators, take into account the width of the operands, results and the intermediate results and use generic definitions and avoid unnecessary penalizing silicon occupation.

Practicals: The labs are composed of two sessions:

During the first session, the students choose and implement one block of the elementary operation which is the dilation/erosion, see Fig. 4. They choose the block to implement according to their previous knowledge (beginners or more advanced). They simulate them on the block level.

During the second session, the students put together their blocks and/or are given the missing blocks by the teacher and simulate the application on the top-most level. Finally they place&route the design, upload the bitstream and verify the design in a real FPGA with a camera and a display (see a photograph of the results Fig. 7).

7 Conclusions

The lecture leads the students to think of algorithms in terms of their HW implementation: data flow on the operator level, data widths, data availability, delay compensations and implementation costs.

Two different lab sessions allow to work at both low (logic) level and high (operator) level of the design, allowing the students to simulate at the HDL level as well as the algorithmic level of the application.

The second session allows to put things together, simulate the algorithm, synthesize and download the bitstream into the FPGA. This allows to have a global vision of the entire hardware development chain.

References

[1] ALTERA. Nios II custom instructions user guide. http://www.altera.com.cn, June 2005.

[2] Peter M. Athanas and A. Lynn Abbott. Real-time image processing on a custom computing platform. *Computer*, 28(2):16–24, 1995.

[3] S. Beucher. Méthodes d'analyse des contrastes à l'analyseur de textures. Technical Report CMM N-625, Ecoles des Mines de Paris, 1977.

[4] P. Guermeur. A New FPGA Architecture for Image Processing : CYCLOP. In *EUSIPCO*, pages 1113–1116, september 1998.

[5] Marc Heijligers. XeTaL-II: a low-power multiprocessing simd architecture. *MEDEA+ Design Automation Conference, Germany*, June 2006.

[6] S. Klupsch, M. Ernst, S.A. Huss, M. Rumpf, and R. Strzodka. Real time image processing based on reconfigurable hardware acceleration. *IEEE Workshop Heterogeneous reconfigurable Systems on Chip*, 2002.

[7] F. Lemonnier. *Architecture électronique dédiée aux algorithmes rapides de segmentation basés sur la morphologie mathématique*. PhD thesis, ENSMP, 1996.

[8] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[9] J. Serra and J.C. Klein. The texture analyser. *J. of Microscopy*, 95(2):349–356, 1972.

[10] XESS. *XSA-3S1000 Board Manual*, June 2005.