A Codesign Approach for a High Performance Vehicle Detector

Manoel E. de Lima, Pablo Viana da Silva, Alejandro C. Frery, Edna Barros
Centro de Informática
Universidade Federal de Pernambuco
CP 7851, 50732-970 Recife, PE
BRASIL
mel@cin.ufpe.br

Keywords: image processing, hardware/software codesign, FPGA, vehicle detection, traffic surveillance.

Abstract:

A high performance architecture for an image processing application, a computer vision system for vehicle detection, is proposed in this work. The system aims to detect and count moving vehicles crossing a selected window from a video input captured by a traffic surveillance camera. The system works under a set of design constraints such as communication throughput, processing time and implementation cost on a real-time situation. The vehicle detector attains high performance on a low cost hardware/software platform, even using 2D FIR filtering, histogram thresholding and color space transformations, among others image enhancing techniques. This paper presents the development of the algorithms in a hardware/software DSP platform based on a field programmable gate array architecture, their implementation and the performance analysis on an application with real data.

1 INTRODUCTION

Traffic congestion is one of the most important problems that affect modern cities. This phenomenon spreads out in small, medium-sized towns and cities as well as in urban areas. Therefore, systems for the analysis and prevention of traffic congestion are becoming more and more relevant. Sophisticated techniques have been used to make traffic congestion prevention possible, but they all depend on the detection and counting of vehicles. These tasks rely on computer vision systems that, in turn, require fast digital signal processing (DSP) algorithms.

Traditionally, DSP algorithms are implemented using general-purpose programmable DSP chips for low-rate applications. High performances could also be attained implementing an Application Specific Integrated Circuit (ASIC). However, technological advancements in field programmable gate arrays (FPGA) or complex programmable logic devices (CPLDs) [2,3,4] in the last seventeen years have opened new paths for DSP design engineers. New options to cope with such problems are now available. Platform FPGAs have become components for implementing high-performance DSP systems, especially in digital communications, video, and image processing applications [5]. These devices preserve the high specificity of the ASIC while avoiding its high development cost and its inability to accommodate design modifications after production. Highly adaptable and design flexible, FPGAs provide optimal device utilization through conservation of board space and system power and the ability to implement highly parallel custom signal processing architectures, some important advantages not available in many stand-alone microprocessors and DSP processors.

However, depending on their capacity, FPGAs can be very expensive, and so the solution. On the other hand, software solutions based on general-purpose processors offer flexibility and permit low cost implementations. Nevertheless, as it was said before, in such approach it is the difficult to exploit the parallelism required by high performance DSP applications. This paper presents a real-time system that performs these tasks employing a low-cost hardware/software platform. In the architecture proposed, the trade-off between hardware and software features is taken into account.

Today, computing demands solutions that target aggressive time-to-market windows of opportunities, and rapid prototyping platform based on such architecture, may effectively respond quickly and economically to the emergence of new, more efficient core algorithms. The software component in this approach is a personal computer (PC) and the hardware component, a PCI platform with reconfigurable FPGA component.

Section 2 presents an overview of the system and the details of the image processing modules. Section 3 describes the vehicle detection algorithm, with an analysis of its complexity. Section 4 presents the case study and, finally, Section 5 presents the main conclusions and directions for further works.

2 SYSTEM DESCRIPTION OVERVIEW

Figure 1 depicts the image processing flow: filtering, amplification, modulation, storage and computation. The frames are captured and treated by the image processing module. The result of this processing is then compared with a reference image in order to identify the presence or not of vehicles in the scene. Another approach can be seen in [1].

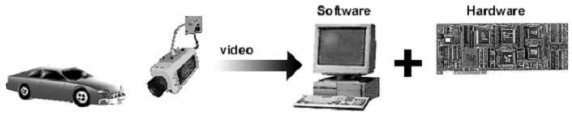

Figure 1. Overview of the computer vision system

The problem is split into processes that are treated as hardware and software components in a codesign approach. The software components are executed in the PC and the hardware ones in the FPGA. The communication among processes is performed via a PCI bus, at 33 MHz, with interrupt signal. The “attached processor” [6], the FPGA

platform, is a XC2S_EVAL board [7], comprising a PCI interface and a 200,000 equivalent gates Xilinx FPGA (Spartan-II Family) [8]. This board also contains a 512kB SRAM memory [9], connected to the FPGA, that stores partial images and parameters during the processing. Figure 2 illustrates hardware platform.

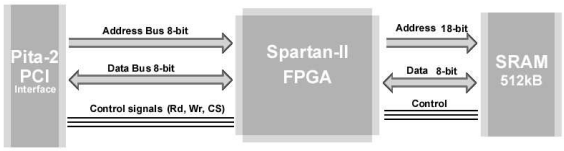

Figure 2. Interconnection between the board elements

The PCI interface is implemented by an ASIC [10]. The physical hardware/software interface is implemented by PCI 8-bits bus. This bus limit reduces the final throughput on host/PCI board interface to approximately 3 MB/sec.

A friendly graphic user interface, shown in Figure 3, was also developed to support the image analysis.

Figure 3. User-friendly visual interface of the system

Through this platform interface the user can select any region for observation, modify the system sensitivity and change the threshold image level. The result, i.e., the vehicle count, is visualized on the screen of the host (the huge "04" in Figure 3).

3 THE VEHICLE DETECTION ALGORITHM

The input to the system is an analog color video stream. After its conversion to a digital color frame [11], the set of primary data is formed by the sequence I_1, I_2, K of color images of the form $I_i : (I_i^R(s), I_i^G(s), I_i^B(s))_{s \in E}$, where the superscript denote the Red, Green and Blue components. Each image is a function $I_i : E \to K$, where the set of coordinates is denoted $E = \{0, \mathrm{K}, m-1\} \times \{0, \mathrm{K}, n-1\}$ and the set of k possible values is $K = [0, \mathrm{K}, k-1]$ (see [12]). In our approach $k = 256$ and the elements of E will be either referred to as (i, j) or as s.

Previous experiences demonstrated that, for the current application, there is enough information in the luminance of the images, so chrominance can be discarded in order to reduce the amount of information to be processed. The luminance Y_i of the color image $I_i = (I_i^R, I_i^G, I_i^B)$ can be computed using the NTSC transmission system definition [13], as

$$Y_i(s) = [0.299 I_i^R(s) + 0.5871 I_i^G(s) + 0.114 I_i^B(s)], \quad (1)$$

where $[x] = \max\{\lambda \in \mathbf{N} : \lambda \le x + 1/2\}$ denotes the closest integer to the real value x. Denote this transformation τ_1; it is a projection of K^3 into K.

Gray scale images (0 as black, $k-1$ as white and intermediate values as intermediate gray levels [12]), produced by τ_1 suffer from noise due to environmental conditions, shadows and vibrations. Thus, they cannot be compared directly to the reference image $I : E \to K$. Simple noise reduction techniques were tested in order to make each gray scale image $(Y_i)_{i \ge i}$ comparable to I. The linear nature of convolution filters made this class of procedures desirable for this application, and noise reduction was sought within the class of low-pass convolution filters.

A kernel, i.e. a matrix $A = [a(i, j)]$, uniquely defines a convolution filter. The size of A is typically very small when compared to that of the image support E. The filtered version of the image Y_i is $F_i = Y_i * A$, defined as

$$F_i(x, y) = \sum_{p,q \in E_A} a(p, q) Y_i(x - p, y - q),$$

where E_A is the support of the kernel A. Borders and corners of Y_i are left unfiltered.

The simplest low-pass convolution filter is the mean, defined by $a(i, j) = \# E_A^{-1}$, where " # " denotes the number of elements. The smallest kernel leading to satisfactory results under several daylight and nighttime conditions and weather situations (sun, rain etc.) was a square of size 3 where $a(i, j) = 1/9$. This reduces the number of required computations for the convolution, since in this case it becomes the simple mean of nine observations around the pixel being processed. Denote this low-pass transformation τ_2:

$$\tau_2(Y_i)(x, y) = \frac{1}{9} \sum_{-1 \le m, n \le 1} Y_i(x - m, y - n).$$

Other low-pass filters may be used as, for instance, those defined by Gaussian kernels, but in our experiences they produced the same or worse results at a higher computational cost.

After the application of τ_1 and τ_2 each color frame is transformed into a filtered gray scale image, suitable for comparison with the reference image I. This is performed with a pixelwise difference followed by an absolute value transformation, i.e., each comparison image $C_i : E \to K$ is given by

$$C_i(s) = \left| F_i(s) - I(s) \right| \quad (2)$$

Denote this transformation τ_3.

In order to reduce the amount of information to be processed, a thresholding was applied to the sequence of comparison images $(C_i)_{i>1}$. This transformation, called τ_4,

produces a sequence of binary images $(B_i)_{i>1}$, $B_i : E \to \{0,1\}$, defined as

$$B_i(s) = \begin{cases} 0 & if \quad Ci(s) < h \\ 1 & if \quad Ci(s) \geq h \end{cases} \tag{3}$$

where h was empirically set to 80, a suitable value under the same conditions used to determine τ_2.

Finally, a summation was performed on the binary images, i.e., each B_i was transformed into an integer $d_i = \sum_{s \in E} B_i(s)$. This value is called *detection index*, since it quantifies the difference between the original color frame I_i and the reference gray level from the image I. Denote this transformation τ_5.

Not every positive value of d_i will be considered as a relevant change. Each original image is composed by 480×640 pixels, but only a small region (E) of 128×256 pixels is surveyed. The detection index d_i is then calculated on this restricted rectangular area, as shown in Figure 3. For this region, and under the aforementioned weather and environmental conditions, values $d_i \geq 10^3$ (threshold) were considered as significant for triggering an event. This value corresponds to typical views of partial or total sections of cars. The value d_i depends on the camera location and the type of target to be detected.

The algorithm searches for local maxima of the sequence $(d_i)_{i>1}$ by comparing d_i with d_{i-1} and with d_{i+1}. If $d_i > d_{i-1}$ and $d_i > d_{i+1}$ then it is assumed that the maximum invasion of the car takes place in image i, and this index is stored.

Figure 4 shows the function $(d_i)_{1<i<34}$, corresponding to approximately 17 seconds of surveillance. The first ten frames show no detection. A car enters in the region under observation in frame 10 and leaves it after three frames. Another vehicle is detected in frame 13, but the peak is smaller since its contrast against the background is reduced. Two big objects are also detected between frames 20 and 22 and frames 29 and 31. No false alarms were detected in the tests and, similarly, every seen vehicle was detected by the system.

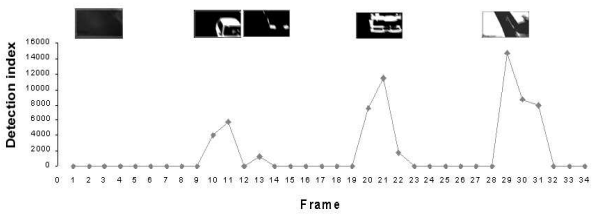

Figure 4. Detection index function

Note that the peaks shown in Figure 4 correspond to their respective detection index.

3.1 Complexity of the image processing algorithm

The choice between hardware or software implementation of the modules that define the image processing algorithm is based on complexity issues, assessed by the number of basic arithmetic operations performed at each stage, such as color conversion, filtering, and so on.

3.1.1 Color conversion

The "RGB to Y" conversion of the images (transformation τ_1) involves pointwise operations, based on equation (1). These operations are composed of three multiplications and two sums for each pixel of the $m \times n$ matrix.

In this work, a rectangular area of 128×256 pixels defines the region of interest within the image being captured. Therefore, the computational complexity regarding to color conversion is given by $128 \times 256 \times 3 = 98{,}304$ multiplications and $128 \times 256 \times 2 = 65{,}536$ sums.

3.1.2 Filtering by convolution

This operation (transformation τ_2) is responsible for most of the operations, being necessary eight sums and one multiplication by fraction (division by nine) for each pixel, except the borders and corners points. Discarding these last elements, the computational cost of this operation amounts to $126 \times 254 \times 8 = 256{,}032$ sums and $126 \times 254 \times 1 = 32{,}004$ products.

3.1.3 Comparison between images

The comparison between each frame and the reference image (transformation τ_3) comprises a subtraction and an absolute value operation for each image element. Considering separately these operations, the system has to compute $126 \times 254 \times 1 = 32{,}004$ subtractions and the same number of absolute value operations. Following equation (3) (transformation τ_4), each result is compared with the thresholding parameter value requiring, thus, $126 \times 254 \times 1 = 32{,}004$ comparisons.

3.1.4 Changing index

The change quantification, that computes the detection index at each frame (transformation τ_5), is obtained by the summation of the binary image pixels. This operation requires $126 \times 254 \times 1 = 32{,}004$ sums.

As the detection index represents the invasion level of an object in the interest area, and the sensibility parameter value discards little intrusions, one comparison with a minimum area is computed for each frame.

3.1.5 Local maxima detection

This operation demands two comparisons at each frame. First between the detection index d_{i-2} and d_{i-1}, after between d_{i-1} and d_i.

3.2 Hardware/software partitioning

In order to reach the processing requirements and so the performance, a hardware/software codesign approach was adopted.

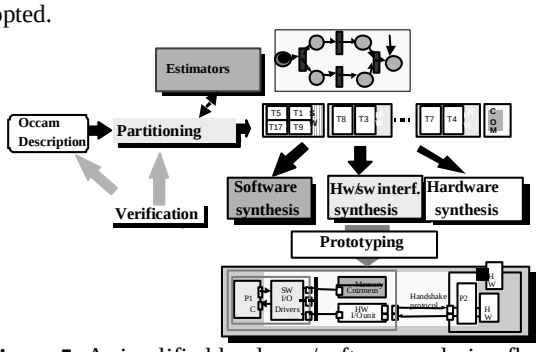

Figure 5. A simplified hardware/software codesign flow.

In this work, the hardware/software approach takes into account the principles of the PISH project [15,16], where the system specification is partitioned into two large blocks, software and hardware, considering aspects related to communication cost between processes and area (Figure 5).

The partitioning of this system is still carried out manually, joining processes in clusters in order to reduce their communication cost.

Control processes or those that do not require intensive computation are implemented in software and the intensive processing processes are implemented in hardware. After partitioning, as shown in Figure 5, processes to be implemented in hardware must be synthesized and the software ones compiled. The interface between hardware and software partitions is also manually generated and synthesized

A K6-II processor, 500 MHZ CPU, plays the role of the software component. The massive processing tasks (arithmetic functions for the convolution and filtering algorithms) and the surrounding peripheral circuitry were implemented in the hardware component, a low cost 200,000 gates Xilinx Spartan-II family FPGA, a PCI interface and a 512K SRAM.

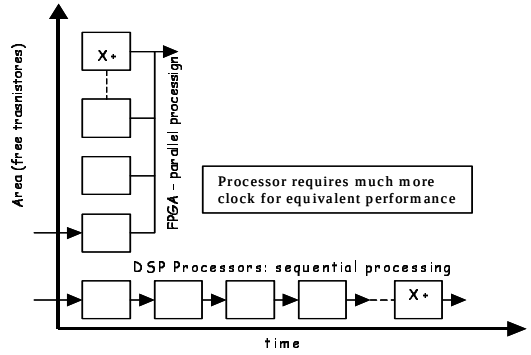

Figure 6: FPGAxDSP and general-purpose processors

When designing a DSP system in a FPGA, the data can be processed taking the advantages of using a single chip, parallel structures (Figure 6) and arithmetic algorithms to exceed the performance of a single general-purpose DSP chip. The designer can take full advantage of the FPGA programmable resources to fit the requirements of any application.

Figure 7 shows the Spartan internal architecture.

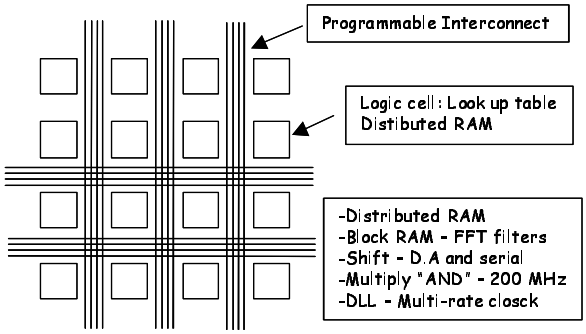

Figure 7: Spartan architecture

The FPGA is configured through the PCI bus through C++ communication classes [17]. Due to the FPGA flexibility the user can easily prototype any algorithm into the entire hardware without the need of a *device-programmer*.

The hardware/software interface is implemented in three layers, by an ASIC, a PCI controller [10], and the C++ class routines in software, as shown in Figure 8.

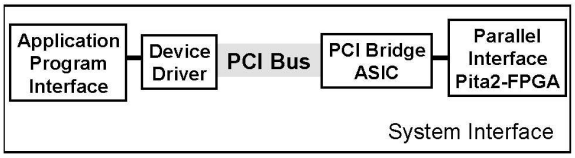

Figure 8. Layers between a hardware device and the software program.

The application program interface represents the portion of the software partition responsible for the communication with the hardware module. It accesses the device driver using C++ classes provided by the prototyping board manufacturer. Then, the device driver addresses the prototyping board and the PCI ASIC connects the PCI Bus with the FPGA through a local parallel interface.

4. CASE STUDY

A real-time processing image application has been developed to validate the algorithm and the codesign approach. One of the requirements is that the system should be able to detect and count intruders in a user-defined region of interest. The intruders have to be detected even if speeding at velocities up to 215 km/h, and their sizes are user-defined.

The images under observation comprise 480×640 pixels, equivalent to 900 kB true-color bitmap files, obtained from a traffic surveillance video.

Initially, a window on the image, free of vehicles and limited to 128×256 pixels, is extracted from the original

image and kept as reference. This reference image is unique for each new scenario and can be taken from any place in the picture scene.

Once captured, each frame is submitted to the image processing algorithm (RGB to grayscale conversion, low pass filter, and so on). As the results of this processing, a signal from the hardware partition indicates the presence or absence of vehicles in that frame depending on the *detection index* presented in section 3.

The system should be able to process at 15 frames/sec, 66.7 ms between frames, allowing an efficient detection of moving objects at high speed (up to 215 km/h). In order to evaluate the advantage of the codesign solution, the image-processing algorithm presented in Figure 9 was also implemented in software.

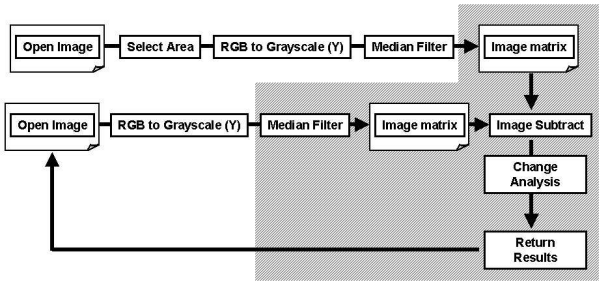

Figure 9. Vehicle detection algorithm at system level description.

The image processing algorithm requires the following steps for each new frame:

$$98{,}304 + 32{,}004 = 130{,}308 \quad \text{Multiplications}$$
$$65{,}536 + 256{,}032 + 32{,}004 = 353{,}572 \quad \text{Sums}$$
$$32{,}004 \quad \text{Subtractions}$$
$$32{,}004 \quad \text{Absolute values}$$
$$32{,}004 + 1 + 2 = 32{,}007 \quad \text{Comparisons}$$

Since the aim of this work is comparing the performance among different architectures, this is a suitable metric rather than the time equivalence for each stage of the processing.

The software version was developed in C++ and compiled using Visual C++ tools, running on a PC with a K6-II processor and 128 MB of SRAM. This software approach was able to process each frame in 220 ms, not fast enough, since only 5 frames/sec could be processed in this period.

In a second approach, a hardware/software co-design methodology was implemented based on the model presented in Section 3.2. The hardware modules, inside the gray box (Figure 9) area cope with massive computation procedures, calculating arithmetic functions and matrix conversions [18]. The software partition is dedicated to the system communication control between the PC and the hardware partition, through writing and reading functions, treatment of the interrupt protocol and user interface. The software partition also performs the reference image fitting (RGB to grayscale conversion and filtering).

The software partition was also implemented in C++, while the hardware one was implemented using VHDL (Very High Speed Integrated Circuit HDL). The FPGA Express tool from Synopsys on Xilinx 3.1i Foundation series was used for hardware synthesis.

Table 1 presents utilization of the reconfigurable logic units of the FPGA.

Logic	Number	% of FPGA
Slices (CLB)	264	11
Flip-Flops	215	4
4-input LUTs	427	9
IOBs	51	36
GCLKs	1	25
GCLKIOBs	1	25
Total equivalent gates	5,086	2.54

Table 1. Logic synthesis report

Five thousand and eighty-six equivalent Xilinx Spartan-II [8] gates were used in this design, amounting to 2.54% of the total available logic on the chip. The maximum circuit frequency reached after synthesis was 54.259 MHz, which satisfies the system clock constraint of 40 MHz imposed by the evaluation board.

The layout of the hardware is depicted in Figure 10.

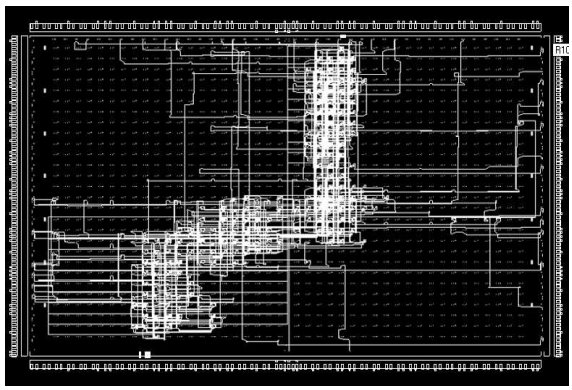

Figure 10. Hardware layout on Xilinx Spartan-FPGA

In this design, the reference image is first sent to the PCI board and stored at a reserved memory. After this, each new frame is also sent to the hardware for processing and analysis. The time spent to transfer each new frame (32kB) from PC to the local memory on the prototyping board is imposed by the 3MB/sec PCI throughput. Thereby, the frame transference usually takes about 10.92 ms.

With the data available at the memory, the FPGA takes around 1,440,180 cycles, equivalent to 36ms (40 MHz), to perform image processing and signaling a response to the software component by an interrupt call. For each processed image, the following mathematics are performed:

$$98{,}304 + 32{,}004 = 130{,}308 \quad \text{Multiplications}$$
$$65{,}536 + 256{,}032 + 32{,}004 = 353{,}572 \quad \text{Sums}$$
$$32{,}004 \quad \text{Subtractions}$$
$$32{,}004 \quad \text{Absolute values}$$
$$32{,}004 + 1 + 2 = 32{,}007 \quad \text{Comparisons}$$

The time analysis expected for each frame processing in this codesign approach is estimated by the sum of the time spent to transfer a image of 32kB (128×256, grayscale 8-bit) from the PC to the prototyping board, plus the internal FPGA processing time. This whole process is performed at 46.92ms for each frame.

Taking into account the area under observation, which depends on the camera visualization angle, and the frame capture rate (15 frames per second), the maximum detectable speed of moving vehicles by this system is of around 215km/h. This complies with the original requirements.

The hardware/software codesign methodology presented in this work shows that good results can be reached in a hybrid approach, analyzing and exploiting peculiarities of each processing element and aspects of their communication.

5. CONCLUSIONS

A versatile reconfigurable DSP platform for image signal processing and analysis in a vehicle detection application has been presented. The optimization of the system, through hardware/software codesign, is guided by the system features. This optimization ultimately leads to a system that, with modest hardware components, meets strict real world requirements.

An affordable prototyping platform was developed and a real case study was presented with encouraging results. Based on a single PC system and a PCI reconfigurable hardware platform, the methodology of synthesis and the prototyping environment provides a good platform that supports the design of reconfigurable systems, for DSP applications, in hardware-on-demand fashion.

The use of the proposed hardware/software codesign approach solved the conflict of timing constraints naturally imposed by the real time characteristics of the problem.

The Spartan filter core is able to produce a fully customized, area-efficient, high performance implementation. Highly optimised FIR filters and others can be fully realized.

In the present version the image processing algorithm partition was performed by hand. However, new applications are under assessment with partitions based on a more formal and automatic methodology.

6. REFERENCES

[1] B. T. C. Jung Soh and M. Wag, "*Analysis of road image sequences for vehicle counting*" Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pp. 679 – 683, October 1995.

[2] S. Hauck, "*The hole of FPGAs in reprogrammable systems*" proceedings of IEEE, vol. 86, pp. 615–638, 1998.

[3] A. DeHon, *Reconfigurable Architecture for General-Purpose Computing*. PhD thesis, MIT, 1996.

[4] S. A. Z. Salsic, *Digital System Design and Prototyping Using Field Programmable Logic*. Kluwer Academic Publishers, 1997.

[5] S. K. Knapp, "*Using programmable logic to acelerate DSP functions*" Xilinx, 1995.

[6] P. M. A. Adit Tarmaster and A. L. Abbott, "*Accelerating image filters using a custom computing machine*" Photonics 1995, 1995.

[7] M. Kraus, *XC2SEVAL User Manual*. Cesys GmbH, Germany, 2001.

[8] *Xilinx Spartan-II 2.5V FPGA Family: Functional Description*, March 2001.

[9] Hitachi Semiconductor, *HM62V8512C Series*, April 2001.

[10] Infineon Technologies, *PCI Interface for Telephony / Data Applications. PITA-2*.

[11] "*XtraConverter: AVI-BMP extractor*." at URL: http://remote-security.co.uk/freebees/xavi2bmp.zip.

[12] G. J. F. Banon, "*Formal introduction to digital image processing*," INPE, S˜ao Jos´e dos Campos, Brazil, July 2000. URL: http://iris.sid.inpe.br:1912/rep/dpi.inpe.br/banon/1998/ 07.02.12.54.

[13] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[14] D. e. a. Demingny, "*How to use high speed recofigurable FPGA for real time image processing.*," Fiftth IEEE International Workshop on Computer Architecture for Machine Perception, 2000.

[15] E. Barros, *Hardware/Software Partitioning Using UNITY*. PhD thesis, Universitaet of Tuebingen, 1993.

[16] C. A. et al, "*Co-synthesis and prototyping suporting the design of reconfigurable systems*" CORE2000: Reconfigurable Computing, pp. 54–67, 2000.

[17] Cesys, *XC2SEval Evaluation Board Programmer's Guide.*

[18] R. S. Stefan Ludwig and S. Singh, "*Implementing PhotoShop filters in Virtex*" Field-Programmable Logic and Applications, Proceedings of the 9th International Workshop, FLP 99, Lecture Notes in Computer Science 1673, pp. 233–242, 1999.