We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

25

Hardware Architectures for Image Processing Acceleration

Almudena Lindoso and Luis Entrena
University Carlos III of Madrid
Spain

1. Introduction

Achieving high performance has traditionally been a challenge in the image processing field. Even though many useful image processing algorithms can be described quite compactly with few operations, these operations must be repeated over large amounts of data and usually demand substantial computational effort. With the rapid evolution of digital imaging technology, the computational load of image processing algorithms is growing due to increasing algorithm complexity and increasing image size. This scenario typically leads to choose a high-end microprocessor for image processing tasks. However, most image processing systems have quite strict requirements in other aspects, such as size, cost, power consumption and time-to-market, that cannot be easily satisfied by just selecting a more powerful microprocessor. Meeting all these requirements is becoming increasingly challenging.

In consumer products, image processing functions are usually implemented by specialized processors, such as Digital Signal Processors (DSPs) or Application Specific Standard Products (ASSPs). However, as image processing complexity increases, DSPs with a large number of parallel units are needed. Such powerful DSPs become expensive and their performance tends to lag behind image processing requirements. On the other hand, ASSPs are inflexible, expensive and time-consuming to develop. The inherent parallelism in image processing suggests the application of High Performance Computing (HPC) technology (Marsh, 2005); (Liu & Prasanna, 1998). As a matter of fact, image processing and computer vision have been the most common areas proposed for the use of HPC. However, actual applications have been few because HPC has failed to satisfy cost, size or power consumption requirements that are usually required in most image processing applications (Webb, 1994).

Hardware acceleration is a suitable way to increase performance by using dedicated hardware architectures that perform parallel processing. With the advent of Field Programmable Gate Array (FPGA) technology, dedicated hardware architectures can be implemented with lower costs. In fact, FPGAs are the cornerstone of Reconfigurable Computing (Todman et al., 2005), a technology that offers unprecedented levels of performance along with a large flexibility. On the one hand, performance can be increased dramatically with the use of custom processing units working in parallel. These units can be mapped to a reconfigurable fabric to obtain the benefits of an application specific approach at the cost of a general purpose product. Cost and time-to-market are also greatly reduced as

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

manufacturing is avoided and substituted by field programming. Finally, power consumption can be reduced as the circuitry is optimized for the application. Reconfigurable computing can also reduce the circuit size (and hence the cost) and provide additional flexibility by run-time reconfiguration. As a result, image processing performance can be improved by several orders of magnitude and becomes affordable for many applications (Memik, 2003); (Diaz et al., 2006); (Gupta et al., 2006); (Bowen & Bouganis, 2008); (Choi et al., 2006).

Modern FPGA devices currently include a large amount of general purpose logic resources, such as Look-Up Tables (LUTs) and registers. In addition, they commonly include an increasing number of specialized components such as embedded multipliers or embedded memories, which are very useful to implement digital signal processing functions. For instance, Xilinx devices include specialized blocks called DSP slices (Xilinx a, 2007). A DSP slice can perform a variety of multiply-accumulate (MAC) operations. Both the DSP-slice operation and the interconnection are programmable.

Many implementations of image processing algorithms on FPGAs have been proposed (Wisdom & Lee, 2007); (Koo et al., 2007); (Note et al., 2006). This chapter describes and analyzes hardware architectures for the acceleration of image processing algorithms. In particular, it focuses on filtering, convolution and correlation techniques, which are very commonly used in many image processing applications. These techniques are a primary target for hardware acceleration, as they are computationally expensive. As a matter of fact, the size of filters and correlation areas must be kept small in many cases in order to avoid introducing large performance penalties. Hardware acceleration can improve the trade-off between performance and complexity, enabling the use of larger filters and correlation areas.

In this chapter, two hardware architectures are described and discussed, based on the spatial domain and on the spectral domain, respectively. Both architectures can be easily parameterized to fit different applications and FPGA sizes. In addition, we show that the approach is not technology dependant and can be implemented in several FPGA technologies. Experimental results demonstrate that performance can be improved by more than two orders of magnitude with respect to a high-end microprocessor.

The integration of the hardware acceleration units with a controlling microprocessor is also studied. As the hardware acceleration units take on the most computationally expensive tasks, microprocessor requirements can be lowered. An architecture based on an embedded microprocessor and a hardware acceleration coprocessor is described. The coprocessor executes the computationally intensive image processing tasks under the control of the microprocessor. This architecture has been optimized for image processing and the hardware coprocessor is able to compute several critical operations for large data sets. The coprocessor architecture exploits the availability of high performance MAC modules in modern FPGAs.

The programming interface of the coprocessor has been optimized in order to reduce configuration time to a minimum. To this purpose, a simple configuration interface built upon the configurable fabric is used instead of the FPGA built-in fine grain configuration mechanism. This way, the performance improvements provided by the coprocessor can be fully exploited while a wide set of typical image processing functions for any image size can be mapped dynamically into the coprocessor.

The proposed coprocessor is dynamically reconfigurable during execution time. Reconfiguration is carried out by the embedded microprocessor, which can specify the

image sizes and the operation to be performed by the coprocessor. As all coprocessor units execute basically the same operation, they can be configured in parallel. This approach combines the benefits of reconfiguration without decreasing performance, as the reconfiguration time has been optimized to complete the reconfiguration process in a few clock cycles. The whole system can be embedded into a single FPGA chip to implement a System on a Programmable Chip (SoPC).
This chapter is organized as follows; in section 2 image processing fundamentals are described, focusing on filtering, convolution, correlation and wavelet transform. Section 3 describes hardware architectures to accelerate the operations described in section 2 in spatial (subsection 3.1) and spectral domains (subsection 3.2). Experimental results obtained with the architectures are summarized in subsection 3.3. Section 4 describes the integration of a hardware coprocessor for image processing in a SoPC. Subsection 4.1 describes the image processing coprocessor and subsection 4.2 summarizes the experimental results obtained with the SoPC. Finally section 5 presents the conclusion of this chapter.

2. Image processing fundamentals

Image processing algorithms involve the repetition of some computations over large amounts of data. We will focus on linear filtering, convolution and correlation techniques, which are very commonly used in many image processing applications. Before going into implementation details, we need to establish the foundation of these techniques.
Linear filtering, convolution and correlation are strongly related concepts. They are typically defined by a sum of products, and can be implemented in a direct form by means of Multiply-Accumulate (MAC) operations. Alternatively, they can be described in the spectral domain and can be implemented using the Fast Fourier Transform (FFT). All these operations introduce a high computational load that is proportional to the size of the images considered.

2.1 Linear filtering, correlation and convolution

Filtering is one of the main techniques used in the field of image processing. Linear filters are a common type of filters that are computed as a linear operation on the pixels of an image. At any point (x,y) in the image, the response G(x, y) of the filter is the sum of products of the filter coefficients T(i,j) and the image pixels overlapped by the filter (Gonzales &Woods,1992):

$$G(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} T(i,j) I(x+i, y+j) \tag{1}$$

The process of moving a filter mask over the image and computing the sum of products at each location is generally called correlation. Corrrelation is a function of the relative displacement (x,y) of the filter T with respect to the image I. Correlation with a discrete unit impulse mask yields a reversed version of the image. Alternatively, we can reverse the filter so that the operation with a discrete unit impulse mask yields the original image. This operation is called convolution and can be expressed as follows:

$$CV(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} T(i,j) I(x-i, y-j) \tag{2}$$

The response of the filter can also be computed in the frequency domain, using the convolution and correlation theorems (Gonzales & Woods, 1992). The convolution theorem states that the Fourier transform of the convolution of two functions is equal to the point-wise product of the Fourier transforms of the two functions. The correlation theorem has a similar formulation, but the product in the frequency domain must be made with the complex conjugate of the Fourier transform of the filter to take reversing into account:

$$CC(x,y) = F^{-1}(F(T(x,y))^{*} * F(I(x,y))) \tag{3}$$

The interest of the correlation theorem lies in the use of the Fast Fourier Transform (FFT) in the previous formula. The complexity of the 2-D Discrete Fourier Transform (DFT) is in the order of $(MN)^2$, but the FFT reduces dramatically the number of multiplications and additions required to the order of $MN\log_2(MN)$. Thus, the correlation in the frequency domain using formula (3) is a good choice for a large filter size.

2.2 ZNCC

Cross-correlation, or simply correlation, is a measure of image similarity that is often used for image matching. In order to compensate differences in brightness, intensity, etc. of the images, a correlation measure that is invariant to affine transformations, such as Zero-Mean Normalized Cross-Correlation (ZNCC), must be used (Gonzales & Woods, 1992), (Crouzil et al., 1996). On the other hand, the measure must take into account the possible relative displacement between the images. Given two images T and I, ZNCC is given by the following formula:

$$ZNCC(p,q) = \frac{CC(T-\overline{T}, I(p,q)-\overline{I}(p,q))}{\left\|T-\overline{T}\right\| \cdot \left\|I(p,q)-\overline{I}(p,q)\right\|} \tag{4}$$

In equation 4, T and I are two images, usually called template and input images, respectively, $\overline{T}$ and $\overline{I}(p,q)$ are the mean values of the respective images and p and q are horizontal and vertical translations of the original input image.

When template images are previously processed, equation 4 can be simplified to equation 5 (Lindoso & Entrena, 2007). In this case, $\overline{T}$ =0 and ||T||= $\sqrt{NM}$, being N and M the dimensions of the considered template image.

$$ZNCC(p,q) = \frac{CC(p,q)}{\sqrt{NMSS(p,q)-S(p,q)^2}} \tag{5}$$

In equation 5, CC is the cross correlation, S is the sum of the input image pixels, SS is the sum of squares of the input image pixels. Formulae 6-8 describe CC, S and SS.

$$CC(p,q) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} T(i,j)I(p+i,q+j) \tag{6}$$

$$S(p,q) = \sum_{i=0}^{i=N-1}\sum_{j=0}^{j=M-1} I(p+i,q+j) \tag{7}$$

$$SS(p,q) = \sum_{i=0}^{i=N-1} \sum_{j=0}^{j=M-1} I(p+i, q+j)^2 \tag{8}$$

2.3 Wavelet transform

The wavelet transform is the cornerstone of multiresolution processing. Wavelet transforms are based on small waves, called wavelets, of varying frequency and limited duration. Wavelet functions can be defined in many ways as long as the set of wavelet functions satisfies some scaling properties (Gonzales & Woods, 2008).
Given a set of wavelet functions

$$\begin{aligned} \varphi_{j,k,l}(x,y) &= 2^{j/2}\varphi(2^j x - k, 2^j y - l) \\ \Psi^{i}_{j,k,l}(x,y) &= 2^{j/2}\Psi^{j}(2^j x - k, 2^j y - l) \end{aligned} \tag{9}$$

the 2-D Discrete Wavelet Function (DWT) of an image I is defined as follows

$$W_{\varphi}(j_0,k,l) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M} \sum_{y=0}^{N} I(x,y)\varphi_{j0,k,l}(x,y) \tag{10}$$

$$W^{i}_{\Psi}(j,k,l) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M} \sum_{y=0}^{N} I(x,y)\Psi^{i}_{j,k,l}(x,y) \qquad \text{i= H, V, D}$$

The results W_{φ} and W^{i}_{Ψ} are called the wavelet coefficients. The coefficients W_{φ} define an approximation of the image I at scale j_0. The coefficients W^{i}_{Ψ} define a detailed approximation of the image I for scales $j > j_0$ in horizontal, vertical or diagonal directions.
The 2-D DWT can be implemented using digital filters and downsamplers. More precisely, the wavelet functions define a low pass filter and a high pass filter that are used to extract the image components in the low and high frequency bands, respectively. The coefficients W_{φ} are obtained by low pass filtering in horizontal and vertical directions (LL filter), while the coefficients W^{H}_{Ψ} , W^{V}_{Ψ} , W^{D}_{Ψ} are obtained by different combinations of low pass and high pass filters, LH, HL, and HH, in order to show details in the horizontal, vertical and diagonal dimensions, respectively.

3. Hardware architectures

The image processing operations summarized in section 2 imply a large quantity of multiply-accumulation (MAC) operations. Microprocessors require several steps (instruction fetching, decoding, execution and write back) for every MAC operation. As these steps are performed sequentially, it is difficult to speed up the computation without architectural changes. To the contrary, FPGAs can provide high performance MAC computation with dedicated architectures that are able to perform several MAC operations at the same time.
In this section two different architectures are described. The architectures perform the computation of cross correlation (equation 6) in the spatial and spectral domains (Lindoso & Entrena, 2007). Other operations such as filtering and convolution can be considered as particular cases of cross-correlation. In the following subsections it will also be described how these architectures can be adapted to perform all the operations described in section 2.

3.1 Spatial architecture

Current FPGAs generally contain a large number of embedded multipliers that can be used for the implementation of MAC operations. A MAC operator module can be built with just a multiplier, an adder and several registers. Besides, modern FPGA families usually include dedicated modules for high performance MAC computation, such as the Xilinx DSP Slice (Xilinx a, 2007), or the Altera's DSP Block. (Altera, 2005).

The spatial architecture consists of a systolic array of MAC modules connected to a memory which contains the images, Figure 1. Inside the array, every MAC module performs the operation and passes the result to the next module in the same row. With this approach, a single data value is read from the input memory at every clock cycle, which is supplied to the first MAC module in each row. The last slice of every row produces the row computation result.

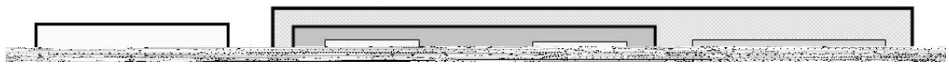

Fig. 1. Spatial architecture

The computation matrix results from the addition of N consecutive row results obtained at different times. This addition is performed by accumulating the row result to the next row. A delay line of N - M cycles is inserted for this purpose between the output of the last MAC module in a row and the input of the first MAC module in the next row, being N and M the size of the rows of I (input image) and T (template or filter coefficients) respectively.

Delay lines can be implemented as shift registers or FIFOs. The choice depends mainly on the selected technology and device, and may affect performance.

The proposed architecture can compute all the operations described in section 2. For CC computation, equation 6, first the template data are stored in the input registers of the MAC modules. After that, the input image is sent to the MAC array to perform the computation. For this purpose, two different data paths (input and template) can be used that may be governed by different clocks. The template path must disable its clock after data loading.

CC operation is quite similar to filtering (the template data can be considered as the filter coefficient matrix) and the computation process is completely equivalent. Convolution is

also possible by reversing one of the images. Wavelet transform can be considered a multiresolution two-dimensional filtering with downsampling. The differences in this case are in the downsampling and the number of iterations required for completing the computation. For this iterative process, intermediate storage (results for each decomposition level) is required. A multiplexer between the MAC array and the memory is also required in order to choose the appropriate input data at every iteration.
This architecture can also compute SS and S as required for ZNCC, equation 5.
The proposed architecture is portable and scalable. Most FPGA technologies include high performance embedded MAC modules in advanced families (Xilinx a, 2007), but multipliers are available in any family and make possible the efficient implementation of MAC modules. The maximum size of the MAC matrix depends mainly on the resources available in the device chosen.
More formally, the correlation/filtering computation can be defined in a recursive manner as it is shown in equation 11:

$$R(i,p,q) = R(i-1,p,q) + \sum_{j=0}^{M-1} T(i,j)I(p+i,q+j) \tag{11}$$

where R(i, p, q) represents the result for the i first rows, R(-1, p, q) = 0 and CC(p, q) = R(N-1, p, q). By making the variable change t = i + p, we obtain:

$$R(i,t,q) = R(i-1,t-1,q) + \sum_{j=0}^{M-1} T(i,j)I(t,q+j) \tag{12}$$

This formula is implemented by the MAC array as follows. At each clock cycle, a new pixel I(t, q) is read from the memory, where the index t is incremented when the index q has completed a row of the image. The MAC module at position (i, j) has T(i, j) in one of the input registers and operates with the value of I that is also passed to the next MAC module in the row at each clock cycle. Then, at a particular clock cycle, given by t and q, each row is computing R(i, t, q) for a different value of i=0, …, N-1. In other words, several row-operations are computed inside the MAC array at the same time. The result of the previous row, R(i-1, t-1, q), is added up at the beginning of each row. To this purpose, a delay line must be inserted, since R(i-1, t-1, q) was obtained during the computation of the previous row.
This approach allows completing the operation in a single memory pass, provided that the size of the MAC array is equal to the size of the template image. The MAC array can compute size(T) MAC operations in a single clock cycle. Therefore, in this case, the number of clock cycles required to complete all correlation computations is equal to the size of the input image plus the latency of a single row correlation. Usually, the latter is negligible. If the template image is larger than the number of available MAC modules, then the template image can be partitioned and operations are computed partially. For the general case, the total number of clock cycles can be estimated as:

$$\text{Nclk} = \text{size(I)} \times \text{size(T)} \,/\, \text{size(MAC array)} \tag{13}$$

In practice, the main timing bottleneck of this architecture may appear at the memory and the delay lines. FPGA families provide small, fast embedded RAM blocks (Xilinx a, 2004). Larger RAMs and FIFOs can be built by composing RAM blocks, at the expense of

decreasing speed. This problem can be solved by dividing the memory clock frequency by two and doubling the data bus width. With this approach, every memory position stores two data values, corresponding to two consecutive pixels. The same approach can also be used for the delay lines. By doubling the data width of the delay lines, they can work at half the frequency and meet the data rate required by the MAC array.

3.2 Spectral architecture

Correlation can also be computed in the spectral domain, as formulated in equation 3. Computations in the spectral domain may be preferred in microprocessor oriented implementations because the number of MACs is reduced by using the FFT (Fast Fourier Transform). Equation 3 requires the computation of 3 two-dimensional Fourier transforms, two direct and one inverse. Assuming that the template T is stored after calculating its FFT, CC can be determined by computing only two two-dimensional FFTs, one direct and one inverse, and a complex multiplication. The Fourier transforms are implemented by two-dimensional FFT.

Figure 2 shows the architecture for the CC computation in the spectral domain. In this architecture a 1-D FFT array is used instead of a MAC array. The 1-D FFT array performs several 1-D FFT simultaneously.

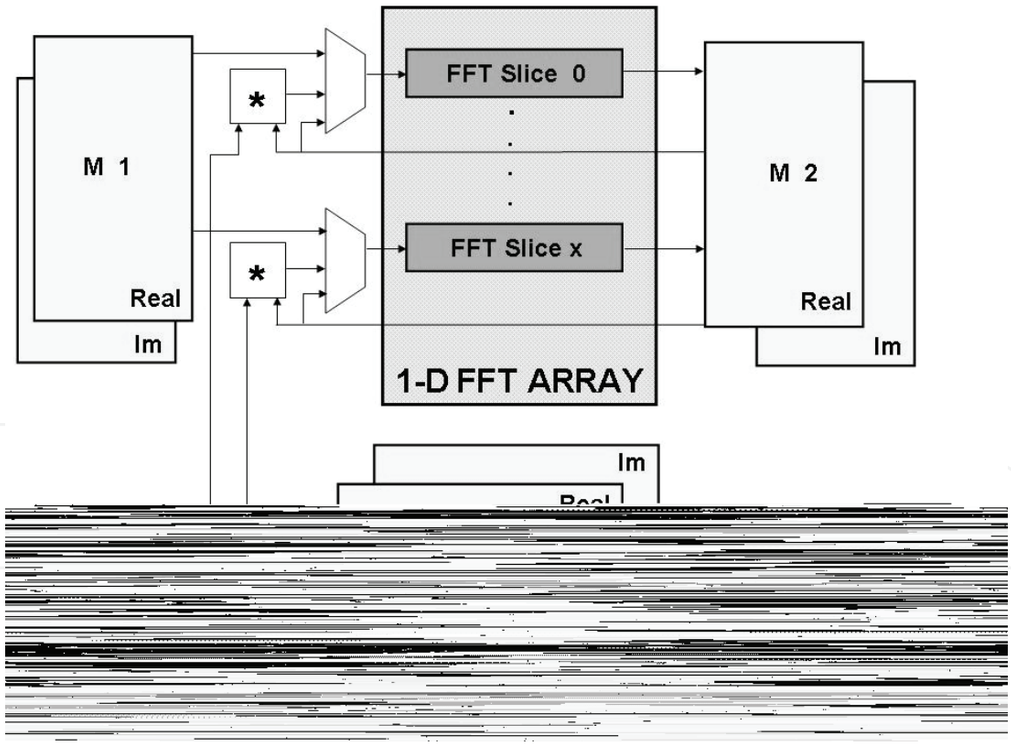

Fig. 2. Spectral correlation architecture

For CC computation, the images are split into rows or columns to perform parallel FFTs. The FFT modules compute the rows/columns FFTs, and are called FFT slices. The maximum number of FFT slices allocated depends on the image size and the device chosen for the implementation.

In this process three different memory blocks are required (Fig. 2): M1 for the storage of the input image and the result, M_T for the storage of FFT(T) and M2 for the storage of intermediate results. All those memories have the same size, which is the double of the input image size in order to store real and imaginary part of the FFT computation. It must be noted that the size of the input image and the template must be a power of 2 in order to compute the FFT. Zero padding is generally used if necessary to meet this requirement. Those three memories can be implemented with internal FPGA memory blocks.
In this architecture the computation of the two two-dimensional FFTs requires four iterations:

1. Compute the FFT of the rows of M1 and store the result in M2.
2. Compute the FFT of the columns of M2 and store the result in M1. After this iteration M1 stores FFT(I).
3. Compute the inverse FFT of the rows of M1 multiplied by the conjugate of the corresponding elements of M_T, and store the result in M2.
4. Compute the inverse FFT of the columns of M2 and store the result in M1.

Since every FFT slice consumes a large amount of resources, in most cases it will not be possible to allocate the number of FFT slices required to complete each iteration in one single step. The maximum number of FFT slices depends on the device available resources and is usually much smaller than the image size.
The memories M1 and M2 are divided into independent blocks in order to provide enough memory ports for each FFT slice. In order to avoid memory access collisions, memories are accessed diagonally by delaying the FFT slices with respect to each other.
The described architecture is useful for convolution and filtering in the spectral domain, and require just minor changes. In particular, convolution implementation is achieved by removing the conjugate operation at the output of M_T.

3.3 Experimental results

In this section the results obtained for several implementations of the architectures described in subsections 3.1 and 3.2 are analyzed. Only CC computation is reported, since other computations produce similar results.
FPGA and software implementations have been considered in order to stablish the speed up achieved with the proposed architectures. Experiments have been conducted with FPGAs from two different FPGA suppliers, namely Xilinx and Altera, in order to test the scalability and portability of the approach.
For the first set of experiments we selected a Xilinx Virtex 4 FPGA, namely XC4SX55-11 with 512 DSP slices. Xilinx V4 FPGA provides embeded high performance MAC modules called DSP slices (Xilinx a, 2007), that can be used for the spatial architecture and also to implement FFT slices in the spectral architecture. The SX subfamily is oriented to digital signal processing applications and contains a large number of DSP slices. Each DSP slice is able to perform many variations of MAC operations at very high speed (up to 500 MHz).
The image size considered is 256x256 pixels. For the device chosen, the maximum number of MAC modules in the spatial architecture is 512 and the maximum number of FFT slices in the spectral architecture is 16. The Fast Fourier Transform Core (Xilinx b, 2004) with radix 4 burst-I/O configuration has been used for the implementation of the FFT slices of the spectral architecture, computing 256 points FFTs.
The experimental results obtained with the Virtex 4 FPGA are summarized in Table 1.

Implementation	SW T_{CC} (ms)	HW T_{clk} (ns)	HW T_{CC} (ms)	Speed Up SW spatial	Speed Up SW spectral
Spatial T=12x12	62	2.477	0.162	240	383
Spatial T=16x16	110	2.438	0.160	244	688
Spatial T=20x20	172	2.490	0.163	239	1055
Spectral (16 FFT slices)	39	4.069	0.145	270	760

Table 1. Xilinx Virtex 4 performance results (spatial and spectral architectures).

Table 1 shows the performance measurements for the architectures and the speed-up achieved with respect to software implementations. For the spatial architecture, three different template sizes have been evaluated (12x12, 16x16, 20x20) as spatial architecture performance depends on the template size. For the spectral architecture only one implementation has been considered (16 FFT slices) because spectral architecture is not dependent on the template size as long as the template is smaller than the input image.

The second column of Table 1 shows the time required by the software implementations (SW Tcc). The reported SW Tcc time has been obtained in a PC equipped with a 3GHz Pentium IV processor and 1 Gbyte of memory. The third and fourth columns of Table 1 show respectively the minimum clock period (Tclk) and the time required to complete the computation of CC of the two images (HW Tcc) with the corresponding FPGA implementation. Finally, the last two columns of Table 1 show the speed-up of the proposed architectures with respect to the software implementations: spectral SW implementation (SW spectral) and spatial SW implementation (SW spatial).

The FPGA implementations achieve a remarkable speed-up of more than two orders of magnitude with respect to software implementations, as it is shown in the last two columns of Table 1. For instance, the spatial FPGA implementation for a 16x16 template performs 688 times faster than an equivalent software implementation and 244 times faster than a spectral implementation in a modern personal computer without loss of accuracy.

Analyzing the resource consumption, the spatial architecture has moderate resource consumption that increases with template size. The main limitation in the spatial architecture is the number of DSP slices available in the device chosen. Larger templates, such as 32x32, need to be partitioned, multiplying the processing time by the number of partitions. In any case, there is enough memory in the device to store the input image and implement the delay lines.

On the other hand, the spectral architecture consumes most of the FPGA resources, including logic, embedded memories and DSP slices, and is only slightly faster than the spatial implementation. The improvement is smaller than initially expected due to several reasons. First, the number of FFT slices that can be allocated is limited by FPGA resources. Second, the minimum clock period is larger than in the spatial architecture, as the critical

delays are in the logic, rather than in other architectural elements. On the other hand, the spectral architecture must be applied on an image size that is a power of two and may be affected by rounding errors. If the input image is to be correlated with several templates (or several rotations of the same template), a large amount of memory is required, since the FFT of each template must be stored. Notwithstanding, the spectral architecture can be advantageous for large template sizes, as the processing time is independent of the template size.

In general, the spectral architecture is less versatile than the spatial one: the range of operations that can be implemented is reduced and the required resources are largely increased. The high amount of required resources provoques that the number of simultaneous FFT slices is considerably small in comparison with the images sizes. In this scenario, several iterations are required to perform a single operation, reducing considerably the speed-up.

The second set of experiments was performed with larger images. In this case we selected a ProcStar II board (Gidel, 2007) with 4 FPGAs. Each FPGA was an Altera Stratix II (Altera, 2007), namely EP2S60-5, and has on-board DDRII memory attached to it to store input and output data. The Altera Stratix II family is not oriented to digital signal processing applications. Each FPGA can implement up to 288 8-bit multipliers that can run at 450 MHz. The selection of a different technology in this case is just to prove the portability and scalability of the described architectures. Both sets of experiments can be performed on each technology and appropriate boards for them are commercially available.

The results obtained in these experiments are summarized in Table 2. In this case the template sizes are smaller since there are less multipliers available. However, the input images are much bigger. As a matter of fact, they must be stored in on-board memory, as they do not fit inside the FPGAs. Only spatial architecture has been considered. The first two columns show the template and input image combinations for each reported experiment. The third column shows the time required by a pure software implementation

Template Size	Input Image size	SW T_{CC} (ms)	HW T_{CC} (Theoretical) (ms)	HW T_{CC} (Experimental) (ms)	Speed up
8x12	1000x1000	631	1,90	3,5	179
8x12	2200X1000	1388	4,19	6,7	209
12x12	1000x1000	944	1,90	3,5	274
12x12	2200X1000	2077	4,19	6,8	307
16x16	1000x1000	1680	1,90	3,1	538
16x16	2200X1000	3695	4,19	7,0	526

Table 2. Altera Stratix II performance results (spatial architecture).

in a 3GHz Pentium IV processor and 1Gbyte of memory. The clock period of the FPGA implementation is 7,613 ns. In this experiment we are using FPGAs with the lowest speed grade that provoques an increase in the clock period with respect to the previous experiment. With this clock period, the fourth column shows the theoretical time required to complete the CC computation. However, in practice the time required is much longer (fifth

column), because of the time needed to transfer the images to the memories and from the memories to the FPGA. Taking all data transfers into account results in as much as 84% time overhead with respect to the theoretical time. Notwithstanding, a speed-up of more than two orders of magnitude is still achieved even though data is stored in external memories and the chosen FPGAs have lower performance.
Experimental results demonstrate that the architecture is portable and scalable given the two FPGA implementations with different image sizes. The architecture can be adapted to any FPGA with or without high performance MAC modules and large speed up over software implementations can be achieved.

4. Image processing SoPC

In the previous section, we have shown hardware architectures for high performance image processing operations. The proposed architectures require an iteraction with other elements (microprocessor or PC) to control the data flow and to perform the rest of the algorithm operations. They can be considered as hardware coprocessors that accelerate a certain operation.
In this section the problem of the interaction with a microprocessor is solved within a System on Programmable Chip (SoPC) scenario. Inside a single FPGA, an embedded microprocessor is connected to a hardware coprocessor that accelerates the most complex operations. The analysis applies also to the case where a stand-alone microprocessor is used. However, SoPC presents advantages because the whole system is located in a single chip, decreasing size, cost and possibly power consumption.
A SoPC design requires an initial device choice in order to adapt the system to the avalaible embedded resources. It must be noted that existing FPGAs present similar options for communication buses and microprocessors. For this case study of a real system, a Xilinx Virtex 5 FPGA, XC5VSX50T, (Xilinx b, 2007) has been chosen. The system is intended for high performance image processing and consists of an embedded microprocessor, a hardware coprocessor that accelerates the critical operations, memories and communication buses between the elements.
Xilinx presents a range of posibilities for the embedded components (microprocessor and communication buses). In this case study, the architecture chosen is shown in Figure 3.
The main components of the system are the following (Fig.3):

- Microblaze soft microprocessor (Xilinx c, 2007), 32 bits wide, 100 MHz, that controls all the system elements, including coprocessor control and data interchange.
- OPB Bus. This is the main bus that is also used to interact with the coprocessor. Data interchange with the coprocessor is made through Direct Memory Access (DMA).
- Coprocessor, which is used for the most computationally expensive tasks.

The main component of the SoPC is the hardware coprocessor that makes possible the acceleration of image processing tasks. The image processing coprocessor is described in the following subsection.

4.1 Image processing coprocessor

The most common image processing operations require a large amount of MAC operations that can be executed in parallel to improve performance. The coprocessor has been designed using the spatial architecture, described in section 3.1, because its versatility along with smaller resources consumption.

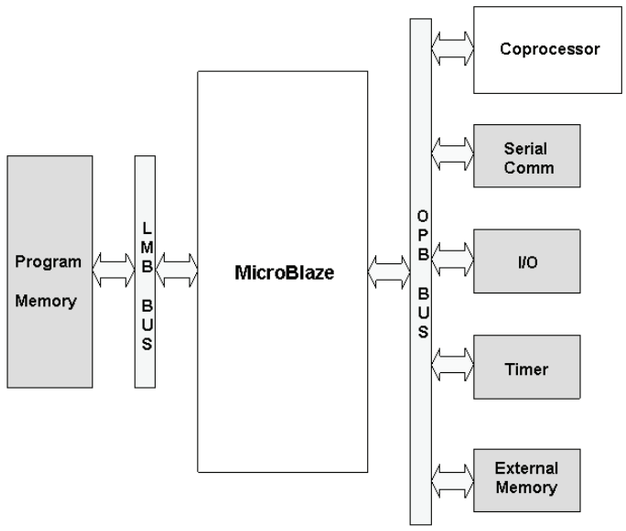

Fig. 3. SoPC architecture.

In order to broaden the number of operations performed, the coprocessor is dynamically reconfigurable. When the coprocessor is reconfigured, the architecture changes to fit the needs of the operation.

Reconfigurable architectures can be classified according to its reconfiguration depth. In this case, fine grain and coarse grain reconfiguration approaches can be considered. FPGAs support a fine grain reconfiguration mechanism. Fine grain reconfiguration provides higher versatility but typically requires large reconfiguration time, even in the case partial reconfiguration is used. On the other hand, coarse grain reconfiguration reduces the versatility but also reduces the reconfiguration time. Thus, an appropriate configuration depth balance must be found in order to achieve the best performance. For image processing purposes, coarse grain reconfiguration is much more efficient since the reconfiguration needs are actually small.

The configuration interface has been compacted to its maximum degree and it is acomplished directly by the embedded microprocessor with 4 transactions of 32 bits. In this approach all processing elements are configured to perform the same operation (MAC operation). The configuration interface can be extended, for instance, by configuring different operations at the processing elements. It must be noted that filter or template data are not considered part of the configuration data.

The coprocessor architecture is shown in Fig. 4. The basic elements of the architecture are: the MAC array, the control block and two I/O FIFOs for data interchange with the microprocessor through the OPB.

The control block manages the I/O FIFOs, the data transfer between the FIFOs and the MAC array, and the operation of the MAC modules. The control block also controls the clocks for the different elements in the coprocessor. In particular, the coprocessor must handle two

data streams that use different clocks. The main data stream is used for image data and is typically controlled by a non-stopping clock. The secondary data stream is used to load filter coefficients or a correlation template. For the latter case, the clock is stopped when all the coefficients have been loaded in order to freeze the contents of the MAC array registers. This operation is required when a filter is computed in order to store the coefficients while the image data is passing through the MAC array.

The coprocessor contains a set of configuration registers that can be programmed by the microprocessor.

The configuration parameters are the following:

- Operation performed by each MAC module.
- Sliding window size. This parameter sets the MAC array dimensions and the length of delay lines.
- Input data dimensions. This parameter sets the image dimensions over which the operations are performed.
- Downsampling rate. This parameter sets the amount of data that is loaded into the output FIFO.
- Control commands, such as coprocessor reset and start commands

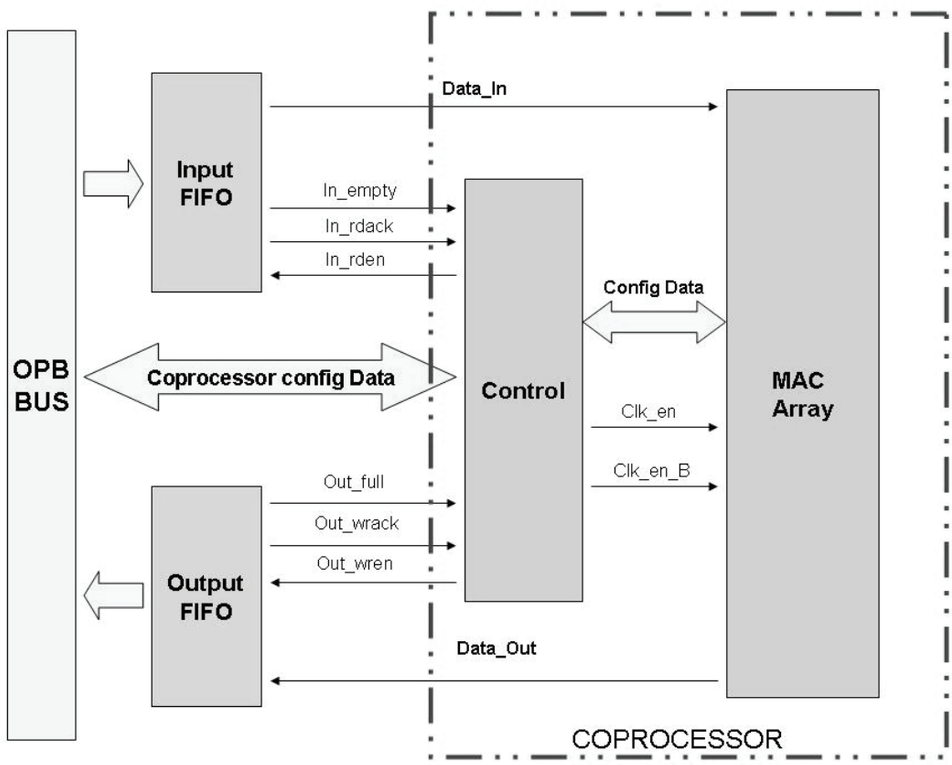

Fig. 4. Coprocessor architecture.

The configuration parameters are packed in 4 32-bit registers, so that full configuration of the coprocessor can be implemented in 4 OPB transactions. Once the coprocessor is programmed, data processing starts immediately as long as data is available at the input FIFO. The coprocessor works in parallel with the FIFO and stores output data at the output

FIFO. Input/output data are transferred from/to main memory by DMA without the intervention of the microprocessor.
The system has been validated with two types of image processing algorithms, described in section 2, which in turn require several coprocessor reconfigurations:

- Wavelet transform. This algorithm involves several two-dimensional filters of the image.
- Zero Mean Normalized Cross-Correlation (ZNCC). This algorithm involves cross-correlation as well as normalization. To this purpose, several reconfigurations of the coprocessor are required in order to compute cross-correlation, sum and sum of squares for sliding windows of the image.

4.2 Coprocessor operations

4.2.1 Wavelet computation

The dyadic wavelet transform involves spatial/spectral filtering that is performed by filtering and subsampling iteratively. The number of required iterations depends on the image size and wavelet level.
For this case study, two wavelet levels are computed. Usually wavelet is computed by detaching horizontal and vertical filters. However, two-dimensional filters are preferred as they can be processed faster by the coprocessor. On the other hand, two-dimensional filters are a common case in image processing applications. In this architecture, 4 two-dimensional filters are considered instead of 8 one-dimensional filters.
Filter coefficients and data are sent to the coprocessor in order to complete the computation. The first level of dyadic wavelet transform requires 4 different two dimensional filters computed over the whole image. Filters are computed and sent to the microprocessor in sequence.
To compute the second wavelet level, the input image is the LL filter result of the first level wavelet transform. This architecture could be optimized by computing several filters at the same time. This can increase performance but the data transfer will be penalized. In this architecture no additional elements are included for data transfer. Once the results are ready they are stored in the FIFO and sent to the microprocessor via OPB. If additional elements are included in order to store data and send it more efficiently, there will be a performance decrease and a connection penalty inside the coprocessor architecture. Adding additional connections can cause a loss in reconfiguration possibilities and a performance decrease inside the coprocessor for a considerable set of coprocessor operations.

4.2.2 ZNCC computation

CC, S and SS (equations 6-8) are computed separately by the coprocessor. Thus, the computation of ZNCC involves 3 reconfigurations. The relative displacement (p,q) that provides the maximum correlation is computed on the microprocessor by traversing the ZNCC results.
For all these operations, the MAC module array uses 4 rows of 48 MAC modules connected in pipeline. The horizontal dimension exploits the MAC column performance to its maximum degree. The vertical dimension has been fixed as the maximum available power of two in order to reduce the computational effort of the microprocessor. It must be noted that the image size is usually larger than the MAC array size. Then, partial results are accumulated by the microprocessor in order to complete the computation correctly.

4.3 Experimental results

The reconfigurable coprocessor performance has been measured for the proposed algorithms. Results were obtained with and without the coprocessor for comparison. In addition, software results were obtained with a PC Pentium IV, 3.2 GHz and 2 GB of RAM.

The coprocessor resource consumption is mainly driven by the MAC array size and the length of the delay lines. FPGA resource consumption is moderate. Half of the FPGA is still empty, so that more components could be added to the SoPC.

Accuracy of the SoPC results and the software results were analyzed without detecting accuracy loss in the considered algorithms.

The operating frequency of the SoPC is 100 MHz, which is Microblaze and OPB frequency.

Performance has been measured for the proposed algorithms. These measurements have been taken for the SoPC, Microblaze without coprocessor and a PC. Different image sizes and wavelet filter sizes have been considered.

Table 3 shows the performance results for the wavelet transform. The wavelet family used is Daubechies 2. These coefficients have been scaled in order to make efficient hardware computations. The results can be correctly rescaled by the microprocessor.

Table 3 shows that the SoPC is working much faster than Microblaze (up to 49 times faster) and slightly faster than the PC. Results show that the achieved speedup mostly depends on the MAC array size, with a clear trend to improve the speedup factor for larger array sizes.

For the two considered array sizes, no additional operations should be performed over the results because the whole two dimensional filters fit inside the coprocessor. If the considered filter size is larger than the available MAC array, the coprocessor would produce partial results and the final results must be computed by Microblaze. In spite of the penalty introduced in such a case by Microblaze computations, the overall SoPC performance will be highly increased because the speedup produced by a larger array size compensates this penalty.

Array Size	Image size	SoPC (ms)	Microblaze (ms)	PC (ms)
4x4	220x220	2.32	112.78	2.58
4x4	110x110	0.59	27.68	0.64
2x2	220x220	2.31	31.02	1.01
2x2	110x110	0.59	7.61	0.26

Table 3. Wavelet transform performance results.

Table 4 shows the performance results for CC. In this case, the SoPC has a much better performance than the other two systems. In particular, for the larger images sizes, the SoPC is 157 times faster than the Microblaze alone and about 3 times faster than the PC. For the considered array sizes the coprocessor is exploiting the MAC modules capacity to its maximum degree. Experimental results demonstrate that a not so powerful microprocessor, such as Microblaze, with the help of the proposed coprocessor can achieve better performance than a modern PC with a very powerful microprocessor.

Tables 5 and 6 show the performance results for S and SS, respectively. Results are quite similar to the results of Table 4. It can be seen that the SoPC performance is the same for the three computations, while the Microblaze and the PC reduce the time for SS and S. Thus, the speedup is proportional to the complexity of the operation.

Note that for CC, S and SS, the coprocessor can only be used for partial computations, as the maximum array size is 4x48. Also, in Tables 4, 5 and 6 it must be noted that when the array size is reduced to the half, the SoPC acceleration is not reduced in the same proportion. The reason is that the number of partial results also decreases and the effort performed by Microblaze is also reduced.

T size	I size	SoPC (ms)	Microblaze (ms)	PC (ms)
48x48	220x220	242	38047	684
48x48	110x110	33	5046	95
24x24	220x220	143	12405	223
24x24	110x110	28	2419	48

Table 4. CC performance results.

T size	I size	SoPC (ms)	Microblaze (ms)	PC (ms)
48x48	220x220	242	19813	634
48x48	110x110	33	2627	87
24x24	220x220	143	6460	214
24x24	110x110	28	1260	42

Table 5. S performance results.

T size	I size	SoPC (ms)	Microblaze (ms)	PC (ms)
48x48	220x220	242	21591	658
48x48	110x110	33	2863	94
24x24	220x220	143	7034	222
24x24	110x110	28	1370	47

Table 6. SS performance results.

Another experiment was performed with the SoPC, testing the performance for ZNCC for images of 560x400 pixels. ZNCC is computed using 3 coprocessor reconfigurations (CC, S, SS) and composing ZNCC result in the embedded microprocessor. Table 7 shows the performance results for SoPC, Microblaze without coprocessor and PC implementations.

Table 7 shows that SoPC reduces considerably the time required by Microblaze to accomplish the computation. Comparing SoPC and PC performance, a speed-up is observed for large templates. It must be noted that most of the time required for ZNCC computation is consumed by the tasks realized by Microblaze. Actually, when T=48x48, from the 632 ms required: CC takes 100 ms, S 87 ms and SS 87 ms, and the rest of the tasks performed by

Microblaze take 358 ms. When T=24x24, Microblaze time increases to 535 ms while the rest of the operations time is reduced slightly (CC=72, S=60, SS=60). In fact the low profile of the embedded microprocessor affects to the achieved speed-up over PC implementations. It must be noted that the performance with a SoPC containing a low profile embedded microprocessor can be even better that a PC Pentium IV.

T size	I size	SoPC (ms)	Microblaze (ms)	PC (ms)
48x48	560x400	632	28549	834
24x24	560x400	727	12166	392

Table 7. ZNCC performance results

5. Conclusion

Hardware acceleration using FPGAs provides a solution to improve the performance of image processing algorithms. In this chapter, hardware architectures for some of the most typical image processing algorithms, such as filtering, correlation and convolution have been presented. Several architectural options have been evaluated, including spectral or spatial architectures, hardware coprocessors or SoPC. These architectures exploit the increasing availability of specific resources in modern FPGAs, such as embedded multipliers and embedded memories, as well as the capabilities of FPGAs to efficiently implement some particular structures, such as delay lines or FIFOs. Experimental results demonstrate that all the presented solutions achieve up to 3 orders of magnitude speed up over equivalent software implementations. The proposed architectures are adaptable to many applications and to the needs of many image processing systems. The architectures are scalable for any FPGA family and adaptable to any FPGA vendor.

To further exploit hardware acceleration, an image processing reconfigurable hardware coprocessor is presented and integrated in a SoPC. This system can provide the flexibility of having a microprocessor that executes C code and take advantage of FPGA high performance for critical operations. Coarse-grain reconfigurability makes the coprocessor adaptable to many operations and algorithm changes without having a negative impact in the system performance. The approach can be easily extended to support other operations in the coprocessor.

6. References

Altera Corporation,(2005) Stratix Device Handbook: DSP blocks in Stratix & Stratix GX devices.

Altera Corporation, (2007), Stratix II Device Family Data Sheet.

Bobda, C. & Hartenstein R. W. (2007), *Introduction to Reconfigurable Computing Architectures, Algorithms and Applications*, Ed. Springer-Verlag, 2007.

Bowen, O. & Bouganis, C.S., (2008), Real-time image super resolution using an FPGA, *Proceedings International Conference on Field Programmable Logic and Applications*, 2008, FPL '08, pp: 89-94.

Choi, J. S.; Lim J. K.; Nam J. Y.; Ho Ha Y.,(2006) Image capture and storage system for digital TV without degrading image quality, *IEEE Transactions on Consumer Electronics*, vol. 52, Issue 2, pp:467 – 471.

Crouzil, A., Massip-Pailhes, L., Castan, S., (1996), A new correlation criterion based on gradient fields similarity, *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 1, pp: 632 – 636

Diaz, J.; Ros, E.; Mota, S.; Pelayo, F.; Ortigosa, E. M.,(2006), Subpixel motion computing architecture, *IEE Proceedings -Vision, Image and Signal Processing,* Volume 153, Issue 6, pp:869 – 880.

Gidel Ltd., (2007), Gidel ProcStar II Data book.

Gonzales, R. C. & Woods, R. E., (1992), *Digital Image Processing*, Addison-Wesley, Reading, MA, 1992.

Gonzales, R. C. & Woods, R. E., (2008), *Digital Image Processing*, Pearson Prentice Hall, 2008.

Gupta, A. K., Nooshabadi, S., Taubman, D., Dyer, M.,(2006), Realizing Low-Cost High-Throughput General-Purpose Block Encoder for JPEG2000, *IEEE Transactions on Circuits and Systems for Video Technology*, Volume 16, Issue 7, pp: 843 – 858.

Koo, J. J.; Evans, A. C.; Gross, W. J., (2007), Accelerating a Medical 3D Brain MRI Analysis Algorithm using a High-Performance Reconfigurable Computer; *International Conference on Field Programmable Logic and Applications*, FPL 2007., pp:11 – 16.

Lindoso, A. & Entrena, L, (2007), High Performance FPGA-based image correlation, *Journal of Real Time Image Processing*, Vol. 2, Ed. Springer-Verlag, pp: 223-233.

Liu, W. & Prasanna, V.K., (1998) Utilizing the power of high-performance computing, *IEEE Signal Processing Magazine*, Volume 15, Issue 5, Page(s):85 – 100.

Marsh, P., (2005) High performance horizons [high performance computing], *Computing & Control Engineering Journal*, Volume 15, Issue 6, Page(s):42 – 48.

Memik, S.O., Katsaggelos, A.K., Sarrafzadeh, M., (2003),Analysis and FPGA implementation of image restoration under resource constraints, *IEEE Transactions on Computers*, vol. 52, Issue 3, pp: 390 – 399.

Note, J.-B.; Shand, M.; Vuillemin, J.E. ,(2006), Real-Time Video Pixel Matching, *International Conference on Field Programmable Logic and Applications*, FPL '06., pp:1 – 6

Todman, T.J., Constantinides, G.A.; Wilton, S.J.E., Mencer, O., Luk, W., Cheung, P.Y.K. (2005), Reconfigurable computing: architectures and design methods, *IEE Proceedings Computers and Digital Techniques*, vol 152, Issue 2, Mar 2005, pp:193 – 207.

Webb, J.A., (1994), High performance computing in image processing and computer vision, *Pattern Recognition, Proceedings of the 12th IAPR International Conference on Signal Processing*, Vol. 3 - Conference C, Page(s):218 – 222.

Wisdom, M. & Lee, P., (2007), An Efficient Implementation of a 2D DWT on FPGA, *International Conference on Field Programmable Logic and Applications*, FPL 2007., pp:222 – 227

Xilinx Inc., (2004), Block RAM (BRAM) Block (v1.00a), www.xilinx.com.

Xilinx Inc., Xilinx LogiCore, (2004), Fast Fourier Transform v3.0, www.xilinx.com.

Xilinx Inc., (2007), XtremeDSP for Virtex-4 FPGAs User Guide, www.xilinx.com.
Xilinx Inc., (2007), Virtex-5 Family overview: LX, LXT, and SXT Platforms, www.xilinx.com.
Xilinx Inc., (2007), Microblaze Processor Reference Guide, www.xilinx.com.

Image Processing

Edited by Yung-Sheng Chen

ISBN 978-953-307-026-1

Hard cover, 516 pages

Publisher InTech

Published online 01, December, 2009

Published in print edition December, 2009

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

InTech Europe

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

InTech China

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

IntechOpen