Implementation of a Wavelet Transform Architecture for Image Processing

Camille Diou, Lionel Torres, Michel Robert

Implementation of a Wavelet Transform Architecture for Image Processing

Camille DIOU, Lionel TORRES, Michel ROBERT
Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier,
UMR CNRS - Université Montpellier II n°5506
161 Rue Ada, 34392 MONTPELLIER Cedex 5, FRANCE
Phone:(+33) 4-67-41-85-85 Fax:(+33) 4-67-41-85-00
e-mail :<author>@lirmm.fr

Key words: Image compression, integrated processor, lifting scheme, multi-resolution analysis, wavelet transform.

Abstract: The wavelet transform appears to be an efficient tool for image compression. Many works propose an implementation of the pyramid algorithm with some improvement to reduce its treatment time or to increase its performances. However, the pyramid algorithm remains silicon area costly, essentially because of its memory needs, and depending on the size of filters used. This paper proposes a new implementation of the wavelet transform using the lifting scheme. This method proposes many improvements such as in-place calculation, small memory needs, and easy inverse transform.

1. INTRODUCTION

Compression is a necessary step for data transmission or storage. Whereas there are many ways of compressing still images, the JPEG method is the most widely used. On the other hand, the video compression often remains based on the MPEG. The goal of this work is to develop a generic wavelet core that can be used as an *Integrated Processor* in integrated systems for many applications as image/video compression, edge detection, moving objects detection. This paper shows a comparison between the Mallat's pyramid algorithm (PA) [1] and the lifting scheme (LS), a new tool for creating wavelets [7,8,10].

Next, we present the pyramid algorithm's implementation based on filter banks, and our architecture based on the lifting scheme.

2. WAVELET TRANSFORM AND MULTI-RESOLUTION ANALYSIS

The multi-resolution analysis [1] uses two functions to project a signal on two spaces:

- the wavelet function extracts the details (high frequency signal);
- the scaling function keeps the approximation (low frequency signal).

By translating and dilating the wavelet, we can analyse the signal all over the time and at different resolution levels. Different wavelet techniques exist for image processing [18,19] depending on the signal to be analysed. We concentrate our efforts on the pyramid and the lifting scheme algorithms.

2.1. Filter banks and pyramid algorithm (PA)

The relationship between the wavelet multi-resolution analysis and filter banks was first shown by Mallat [1]. The image is filtered by both high-pass and low-pass filters along horizontal direction, giving, respectively, an approximation of the original image and its horizontal details. This scheme is re-applied on the two sub-images along the vertical direction, giving us the three horizontal, vertical and diagonal details sub-images, and the 2-D approximated sub-image (Figure 1, left).

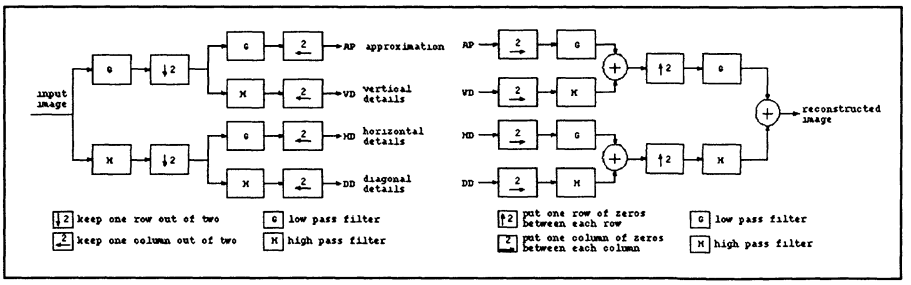

Figure 1. Wavelet transform image decomposition (left) and recomposition (right)

The inverse transform is obtained by inverse filtering with the corresponding filters (Figure 1, right).

2.2. The lifting scheme (LS)

The lifting scheme algorithm [7,8,9,10] presents many advantages compared to the pyramid algorithm [9]:

- calculations are performed in-place for an important memory saving;

- the algorithm shows an inherent SIMD parallelism at all scales;
- inverse transform is obtained easily from the direct transform;
- this scheme can be performed even when the Fourier techniques are no longer suitable (for example when the samples are not evenly placed, that causes problems with the filter banks because of the sub-sampling)

2.2.1. Transform

It is composed of the three following steps (see Figure 2 for details):

- **split**: this step separates the signal into two parts. The more correlated the sub-signals are, the better the predict; practically, the signal is divided into a set of even indexed samples (*S*), and another one of odd indexed samples (*D*);
- **predict**: the first set *S* is used to predict the second one *D*, according to a defined function; the difference between the predicted set and the original one is kept as the detail signal;
- **update**: the detail signal *D* is used to update the unmodified set *S*, in order to keep the average value of the signal.

This can be resumed by the following implementation:

$$(\text{odd}_{j-1}, \text{even}_{j-1}) := \text{Split}(s_j)$$

$$\text{odd}_{j-1} -= \text{Predict}(\text{even}_{j-1})$$

$$\text{even}_{j-1} += \text{Update}(\text{odd}_{j-1})$$

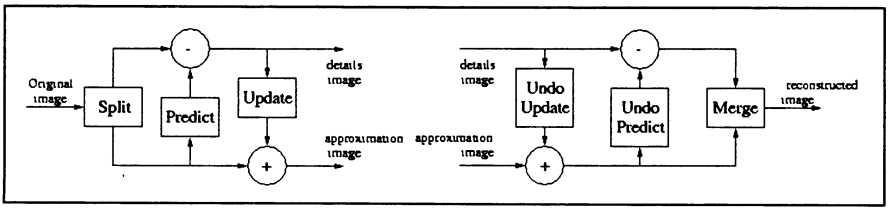

Figure 2. Signal decomposition (left) and recomposition (right) with lifting scheme

Figure 2 gives the schemes for 1-D decomposition-recomposition. For a 2-D transform, we simply apply the same method to each output, and process the scheme along the vertical direction.

The implemented wavelet depends on the prediction function. The higher the degree of the prediction function is, the smoother the wavelet. If the prediction is very close to the signal, the detail coefficients (*i.e.* the wavelet coefficients) will be very small.

Let's take an example with a linear prediction: if the original signal x is split into the detail signal d and the smooth signal s, we have:

Before the prediction:

(1) $d_k = x_{2k+1}$

(2) $s_k = x_{2k}$

After the linear prediction:

(3) $d_k = x_{2k+1} - \frac{1}{2}(x_{2k} + x_{2k+2})$

$s_k = x_{2k}$

And after the update:

$$d_k = x_{2k+1} - \frac{1}{2}(x_{2k} + x_{2k+2})$$

(4) $s_k = x_{2k} + \frac{1}{4}(d_{k-1} + d_k)$

The equation 3 gives us the corresponding high-pass filter, and when inserting equation 3 into 4, we get the corresponding low-pass filter:

$$HP = \left[-\frac{1}{2}; 1; -\frac{1}{2}\right] \qquad LP = \left[-\frac{1}{8}; \frac{1}{4}; \frac{3}{4}; \frac{1}{4}; -\frac{1}{8}\right]$$

2.2.2. Inverse transform

The inverse transform scheme contains three steps (undo update, undo predict and merge) which are obtained by reversing the order of the operations and by changing the signs of the operators, as shown in Figure 2.

We resume it with:

$$\text{even}_{j-1} -= \text{Update}(\text{odd}_{j-1})$$

$$\text{odd}_{j-1} += \text{Predict}(\text{even}_{j-1})$$

$$s_j := \text{Merge}(\text{odd}_{j-1}, \text{even}_{j-1})$$

The equivalent filter is obtained by putting one detail coefficient to 1, all others to 0, and computing the inverse transform. It gives the high-pass filter. Doing the same with a smooth coefficient gives the low-pass filter.

$$HP = \left[-\frac{1}{8}; -\frac{1}{4}; \frac{3}{4}; -\frac{1}{4}; -\frac{1}{8}\right] \qquad LP = \left[\frac{1}{2}; 1; \frac{1}{2}\right]$$

Note that the filters we obtain correspond to the wavelet defined by Cohen, Daubechies and Fauveau (CDF) [2].

3. ARCHITECTURES

There are many works on the implementation of the WT on FPGA [11,12] or ASIC [13,14,15,16,20]. Most of them propose improvements to the pyramid algorithm [14,17], or to the implementation (systolic or semi-systolic architecture, parallel or semi-parallel design [13,15,16]). But these methods rely on the same algorithm to perform the WT.

In this section, we present how we can implement this wavelet transform, first using filter banks, then with the lifting scheme. Then, we compare the methods to point out the number of needed operators, the silicon area cost and the memory cost of each method.

3.1. Filter banks architecture

Figure 3 shows a basic implementation of the pyramid algorithm for a 1-D wavelet transform. The number of operators is chosen in order to correspond to the CDF wavelet (the implementation with lifting scheme is shown in Section 3.2).

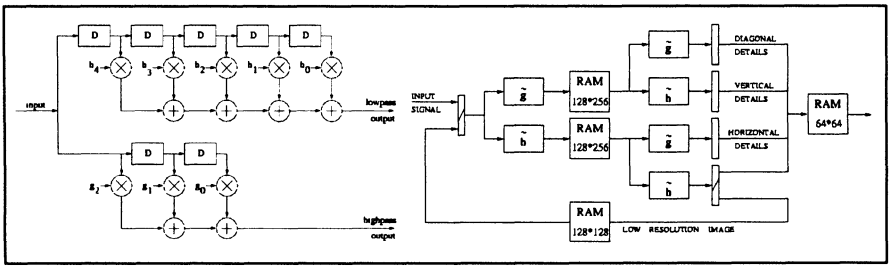

Figure 3. Pyramid algorithm implementation. Left: basic filtering cell. Right: whole structure

An implementation of the 2-D wavelet transform using the CDF wavelet is shown in [11]. We can see, in Figure 3 that the filters described above are present three times and that an important memory is necessary to perform the transform. At least the two 128×256 memories must be on-chip if we want to keep good performance during the transform. All these memories and the filters need an important silicon area. The inverse transform is not shown, but it needs a consequent memory too: 112 kB are necessary in the architecture presented in [11].

Line processing	Column processing		Total
31.7 ms	31.7 ms		64 ms
Data reading Of first cells	Data reading Of last cells	Image writing	Total
4.556 ms	8.857 ms	3.491 ms	17 ms

Table 1. Compression and decompression times of a filter banks architecture

The Table 1 shows the theoretical performances for this system for 5 levels of resolutions on a 256×256 image (see [11]), taking into account the following values: 250 *ns* per pixel for the analysis, and 40 *ns* per pixel for the synthesis. This table does not show the times of the quantization/de-quantization process, which are small compared to the transform.

The evaluated number of CLB for an implementation in a XC4005 family FPGA is about 340. Thus, the implementation of the wavelet transform needs around 5 kgates, without the memory.

3.1.1. Lifting scheme architecture

1-D structure We saw in the section 2.2 that the implementation of the lifting scheme requires few operators. We need, for each step (prediction and update), two adders in the decomposition and two adders in the recomposition. We can see in Figure 4 that there is no need for memory during the 1-D transform: the transformed coefficients overwrite the original ones during the decomposition. Thus the transform block can be seen from outside as a delay line. The transform can be performed in pseudo real-time, depending on the system clock. We get the same conclusion for the reconstruction step.

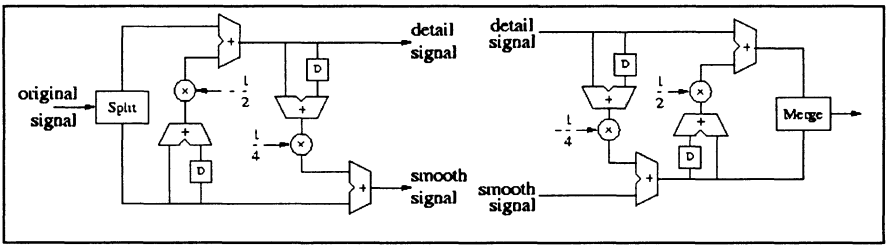

Figure 4. Structure of the lifting scheme blocks: transform (left) and inverse transform (right)

2-D structure For the 2-D implementation, we use the same blocks, organised in a cascade as shown in Figure 5.

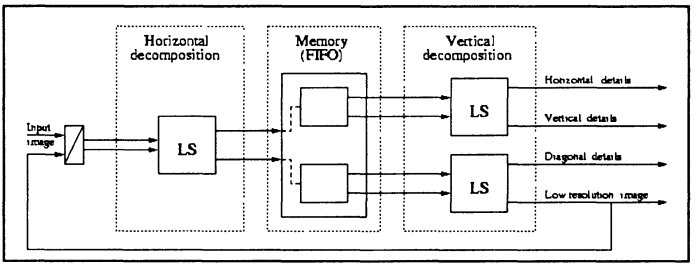

Figure 5. Structure of the lifting scheme block for a 2-D transform

Unlike the 1-D case, we need, in the 2-D case, some memory to perform the transform. Before starting the transform along the columns, we have to wait until the transform along the rows is achieved. There are different ways of doing this:

- we wait that the first 1-D decomposition is achieved, we store the image in a memory, rotate it by 90° or address the memory in a different manner, and we perform the second decomposition;

we only wait that a few rows are treated and then start the vertical decomposition on these rows. When the following rows are achieved, we continue the vertical decomposition on them.

This last case is the most interesting from a memory point of view, but we also have to define the method to access the memory: we receive a few lines, and we want to transform them along the columns.

Figure 6 shows how we can start the vertical decomposition after the horizontal decomposition of the three or four first ones is achieved. We start the horizontal decomposition by computing the detail coefficients, and re-use them to compute the approximation. Once the 3 first lines are treated, we can start the vertical decomposition.

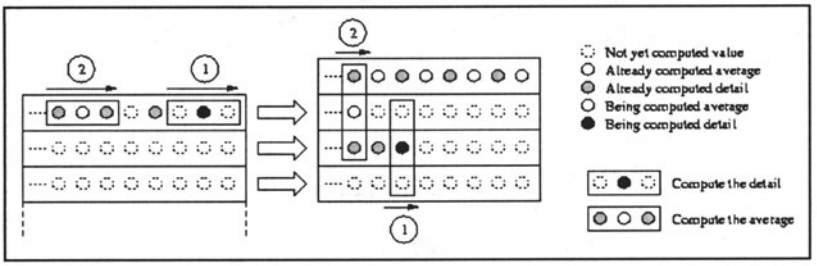

Figure 6. Memory need of the 2-D lifting scheme

When the fourth line (detail) is treated, the line of vertical detail coefficients is achieved too. Both this line and the previously treated one can then be used to compute the vertical average coefficients. While the vertical average is treated, the next vertical detail can be processed concurrently, while the horizontal average is also being calculated. Thus, in order to perform the complete 2-D transform, we only need a memory that can contain 4 lines of the image, *i.e.* for a 256×256 images at 8 bits per pixel, 4×256×8=1 kBytes.

3.2. Comparisons

Number of logic blocks. The lifting is very interesting for integrated systems because of little need for logic blocks. The table below show the difference between the lifting scheme and the filter banks.

Lifting scheme		Filter banks	
Total (2-D)	12 adders	Total (2-D)	21 adders

Table 2. Operators necessary for one step (transform or inverse transform)

We saw that the filter banks architecture needs around 340 logic cells. The lifting scheme only needs around 190 (24 adders × 8 blocks per adder).

Furthermore, to perform the multiplication, using bit shifting, the lifting scheme needs only 12 shifts whereas filter banks needs, for a 1-D transform. For a 2-D transform, this increases to 36 and 102 respectively.

Memory. One of the main advantages of the lifting scheme is the in-place calculation. We don't need any buffer memory to perform the transform: the wavelet coefficients overwrite the original ones (Figure 7). Figure 7 shows, on a real image, the Mallat and lifting scheme coefficients distribution after a 2-level decomposition.

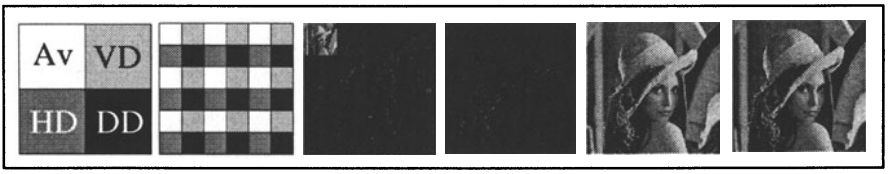

Figure 7. Distribution of the wavelet's coefficients: *a:* pyramid algorithm and *b:* lifting scheme and 2-levels decomposition of an image. *c:* Mallat and *d:* LS coefficients distribution. *e:* original and *f:* re-composed images

The on-chip memory needed for the filter banks design is approximately the size of an entire image whereas, for a lifting scheme implementation, this memory is reduced to a few lines. *Table 3* shows the memory necessary for the filter banks method as described in [11].

	Lifting scheme	Filter banks
Decomposition	1 kB	84 kB
Recomposition	1 kB	112 kB

Table 3. Memory need of the lifting scheme compare to the filter banks for a 256×256 image of 256 grey-levels

Thus, the lifting scheme implementation needs two times less logic blocks and very little memory compared to filter banks architecture. Furthermore, as the lifting scheme computes the wavelet transform in-place, we want to evaluate its performance at video rate. We see, in the next section, a first implementation of the lifting scheme.

4. HARDWARE IMPLEMENTATION

4.1. Overview

We have started to validate the lifting scheme architecture in real-time with an APTIX prototyping platform [21]. This programmable platform (Figure 8) contains Altera 10k100 FPGA, used to implement the wavelet transform with the necessary memory. A DSP core (ST D950) is used to

perform the quantization and the coding of the wavelet coefficients. Figures 10 and 11 show the implementation of the 2-D lifting scheme. The input image is first decomposed horizontally. We get two sub-images at half the pixel clock frequency. Thus, we can alternatively decompose them along the vertical direction, switching between detail sub-image and smoothed sub-image. To keep the video rate, we have to interlace the two sub-images' lines - putting one data of the first sub-image in the memory, and then one data of the second sub-image - and compute the resulting image along vertical direction.

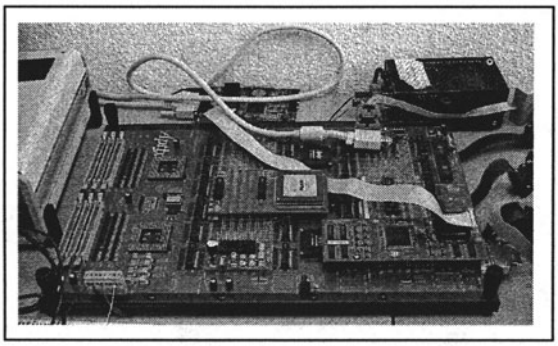

Figure 8. APTIX prototyping platform

4.2. Horizontal decomposition of the input image

Because of the in-place calculation, the horizontal decomposition of the image can be performed at the video rate. The original set of data at the frequency F is decomposed into two subsets (detail and average) at the frequency F/2. A simulation of the 1-D horizontal lifting scheme decomposition shows that this block needs 116 Altera's Flex 10k100 logic cells, that is, 2% of the chip. Thus, there is no noticeable difficulty in implementing the first 1-D horizontal decomposition block. Thus, we'll point out the vertical decomposition below.

4.3. Vertical decomposition of the sub-images

There are different ways of computing the vertical decomposition: we can use FIFO memories or RAM. The FIFO seem to be more efficient because they need no memory managing, in the strict sense of the term. But, the logic necessary to manage the four FIFO increases considerably the complexity. We describe here the two methods.

Using FIFO memories We consider that a first computed odd line DD_{n-1} is present in the first FIFO F1. A first not computed even line S_n is in

the FIFO F2. A first not computed odd line D_{n+1} is in the third FIFO F3. When the second not computed even line S_{n+2} is ready, we compute the detail coefficient line from S_n, D_{n+1} and S_{n+2}. We name these coefficients DD_{n+1}.These coefficients are used to compute the average coefficients SS_n from DD_{n-1}, S_n and DD_{n+1}. The coefficients SS_n and DD_{n-1} are written to the output of the FIFO F2 and F1 respectively. The DD_{n+1} replaces the DD_{n-1} in F1 and the S_{n+2} replace the S_n in F2. Now, we have DD_{n+1} in F1 and S_{n+2} in F2; F3 is free. When D_{n+3} is ready, we stock it in F3 and wait for S_{n+4} which is necessary for calculating DD_{n+3}. Then, the scheme can be repeated from the beginning.

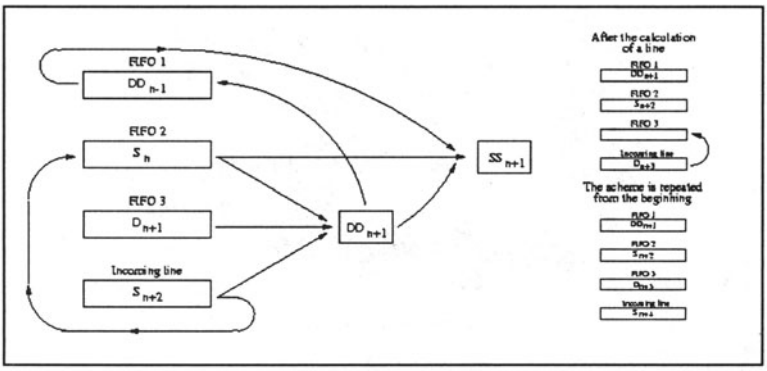

Figure 9. Three FIFO are used to perform the vertical decomposition

Using RAM The method described above shows the complexity of the FIFO memories managing. We have to use numerous multiplexers and demultiplexers for choosing between the video input (the output of the 1D transform block) or the output of the FIFO. All the logic that must be developed could be used to implement a RAM controller. Thus, instead of using FIFO memories, we could use RAM and address the data in a standard way (Figure 11).

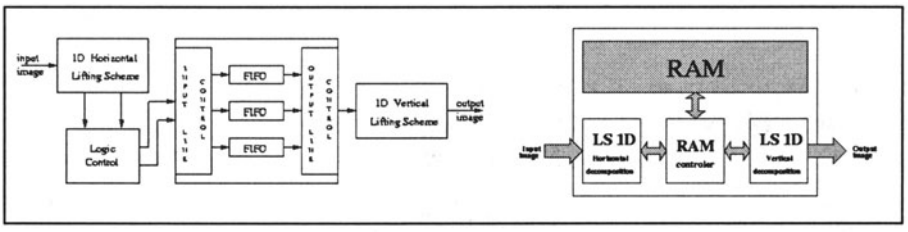

Figure 10. (left) Implementation of the lifting scheme using FIFO
Figure 11. (right) Implementation of the lifting scheme using RAM

4.4. Implementation

A first evaluation of the system presented in Figure 10 shows that the system uses 554 Altera's logic cells (*i.e.* 25% of the Flex 10k100 chip), and 6144 memory bits, that is 25% of the total memory of the Flex 10k100. All these results are given for 8 bits bus width. With a clock frequency about 30 MHz, this will allow the treatment in real-time, of images with a size of 800×800 pixels in 256 grey-level by increasing the size of the FIFO or the RAM. The needed memory can be implemented on the Altera's 10k100 FPGA. By using pipeline techniques, we estimate that we can obtain a clock frequency of 60 MHz, allowing us to process images in HDTV format.

First results of using the LS for image compression show a compression ratio of about 10 for a Peak Signal to Noise Ratio (PSNR) of 30 dB, which is the minimal admitted value for a good image quality. We are currently working on improving the compression ratio to get a value around 40, by using a well-adapted quantization and arithmetic coding [4,5,6].

5. CONCLUSION

In this paper, we have shown a new way of implementing the wavelet transform. This method combines an efficient processing rate with low area cost and memory use. It can easily be adapted to perform the inverse transform. The design re-use and IP cores concept is becoming increasingly present in the top down design flow; it is changing the way electronic engineers works. The impact on time-to-market can be considerable; for this reason, we intend to develop a complete architecture for wavelet transform which can be used as a wavelet IP core for image processing applications.

6. REFERENCE

[1] Stéphane G. Mallat, "A Theory for Multi-Resolution Signal Decomposition: The Wavelet Representation", *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 11, N° 7, July 1989

[2] A. Cohen, I. Daubechies and J. Feauveau, "Bi-orthogonal bases of compactly supported wavelets", *Comm. Pure Appl. Math.*, vol. 45, pp. 485-560, 1992

[3] Chok-Ki Chan and Chi-Kit Ma, "A Fast Method of Designing Better Codebooks for Image Vector Quantization", *IEEE Transactions on Communications*, vol. 42, N° 2/3/4, February-March-April 1994

[4] Marc Antonini, Michel Barlaud, Pierre Mathieu and Ingrid Daubechies, "Image Coding Using Vector Quantization in the Wavelet Transform Domain", *Proceedings IEEE ICASSP*, pp.2297-2300, April 1990

[5] Marc Antonini, Michel Barlaud, Pierre Mathieu and Ingrid Daubechies, "Image Coding Using Wavelet Transform", *IEEE Transactions on Image Processing*, vol. 1, N° 2, April 1992

[6] Pierre Mathieu, Michel Barlaud and Marc Antonini, "Compression d'image par transformée en ondelette et quantification vectorielle"

[7] Wim Sweldens, "The Lifting Scheme: A Custom-design Construction of Biorthogonal Wavelets", *Appl. Comput. Harmon. Anal.*, vol. 3, N°2, pp. 186-200, 1996

[8] Wim Sweldens, "The Lifting Scheme: A Construction of Second Generaton Wavelets", Technical Report 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, May 1995

[9] Wim Sweldens and Peter Schröder, "Building your Own Wavelet at Home"

[10] Gabriel Fernandez, Senthil Periaswamy and Wim Sweldens, "LIFTPACK: A Software Package for Wavelet Transforms using Lifting"

[11] Frédéric Truchetet and André Forys, "Implementation of still-image compression-decompression scheme on fpga circuits", *SPIE*, vol. 2669, pp. 66-75, January 1996

[12] Philipe Guermeur, Stéphane Guermeur and André Picaud, "Une implantation de la transformée en ondelettes discrètes à base de circuits FPGA"

[13] Jimmy C. Limqueco and Magdy A. Bayoumi, "A VLSI Architecture for Separable 2-D Discrete Wavelet Transform", *Journal of VLSI Signal Processing*, N° 18, pp. 125-140, 1998

[14] Gauthier Lafruit, Francky Catthoor, Jan P. H. Cornelis and Hugo J. De Man, "An Efficient VLSI Architecture for 2-D Wavelet Image Coding with Novel Image Scan", *IEEE Transactions on Very Large Scale Integretion (VLSI) Systems*, vol. 7, N°1, March 1999

[15] José Fridman and Elias S. Manolakos, "On the synthesis of regular VLSI architectures for the 1-D discrete wavelet transform", *SPIE Conference on Mathematical Imaging: Wavelet Applications in Signal and Image Processing II*, San Diego, July 1994

[16] Aleksander Grzeszczak, Mrinal K. Mandal, Sethuraman Panchanathan and Tet Yeap, "VLSI Implementation of Discrete Wavelet Transform", *IEEE Transactions on Very Large Scale Integretion (VLSI) Systems*, vol. 4, N°4, December 1996

[17] I. Urriza, J.I. Artigas, J.I. Garcia, L.A. Barragan, D. Navarro, "Locality Improvement of the Pyramid Algorithm to Compute the Discrete Wavelet Transform", *Proceedings of XIII Design of Circuits and Integrated Systems Conference*, pp. 30-35, November 1998

[18] B. B. Hubbard, "Ondes et Ondelettes", *Pour la Science*, dif. BELIN, Paris, 1995

[19] Frédéric Truchetet, "Ondelettes pour le signal numérique", *Collection traitement du signal*, HERMES, 1998

[20] Analog Devices, "Ultralow Cost Video Codec", *Datasheet*, 1997, http://products.analog.com

[21] APTIX, http://www.aptix.com