

# A Scalable and Programmable Architecture for 2-D DWT Decoding

Massimo Ravasi, Livio Tenze, and Marco Mattavelli

**Abstract**—The compression of still images by means of the discrete wavelet transform (DWT), adopted in the JPEG-2000 and MPEG-4 standards, is becoming more and more widespread because it yields better performances than other compression methods, such as discrete cosine transform. The demand of efficient architectures for 2-D DWT coding and decoding for a variety of different applications and embedded systems is rapidly increasing. This paper presents the implementation of a 2-D DWT decoder for Mallat-tree decomposition, suitable for low power applications, such as portable devices. The decoder design has been synthesized and validated in 0.35- $\mu\text{m}$  CMOS technology. The architecture is scalable according to the desired maximum image size, the maximum DWT kernel length, and arithmetic accuracy, and it is programmable at run-time to process different image sizes and use different DWT kernels.

**Index Terms**—Digital signal processors, discrete transforms, image coding, wavelet transforms.

## I. INTRODUCTION

**T**EXTURE coding based on wavelet transform is playing a leading role for its higher performances in terms of signal analysis, multiresolution features, and improved compression compared to existing methods such as the discrete cosine transform (DCT)-based compression schemes adopted in the old JPEG standard. This success is testified to by the fact that the wavelet transform has now been adopted by MPEG-4 for still texture coding and by JPEG-2000. Indeed, superior performance at low bit-rates and transmission of data according to client display resolution are particularly interesting for mobile applications. The wavelet transform shows better results because, thanks to its time-scale representation, it is intrinsically well-suited to nonstationary signal analysis, such as images [10]. Although it is a rather simple transform, discrete wavelet transform (DWT) implementations may lead to critical requirements in terms of memory size and bandwidth, possibly yielding costly implementations. Extended state-of-the-art researches showed that DWT coding and decoding algorithms can be redesigned by changing the scheduling of operations, yielding more efficient implementations with reduced memory requirements [3], [4], [6]. Further work proposed a variety of strategies, dealing with the tradeoff among implementation complexity, cache memory requirements, and external memory requirements [1], [2]. Thus, efficient implementations must be

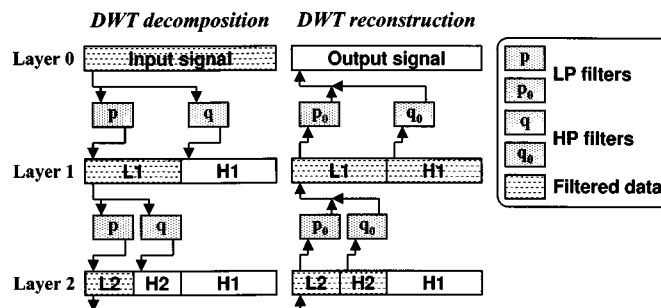


Fig. 1. 1-D DWT with Mallat-tree decomposition. The number of samples of the encoded signal is equal to that of the input signal.

investigated to fit different system scenarios. In other words, the goal is to find different architectures, each of them specifically optimized for any specific system requirement in terms of complexity and memory bandwidth.

Because the iterative subband decomposition is the core process of wavelet transforms, the coding/decoding stage has to be performed on several layers, as shown in Fig. 1, in the case of 1-D Mallat-tree decomposition, where only approximation signals ( $Lx$ ) are recursively split into two subsignals. Along the tree decomposition, it can be noticed that in intermediate layers some data are just temporary. For instance, in layer 1 represented in Fig. 1,  $L1$  signal data is produced while coding input signal and is successively split into  $L2$  and  $H2$ . Similarly, such temporary data can be found also in the reconstruction process.

2-D DWT coding is usually based on separable basic scaling functions and wavelet bases so that it can be performed iterating two orthogonal 1-D DWT. This fact implies the presence of additional temporary samples between horizontal and vertical processing. As shown in Fig. 2, for the 2-D DWT reconstruction, not only the temporary signal  $LL1$  is needed, produced by decoding layer 2, but also temporary signals  $L1$  and  $H1$ , produced as result of horizontal processing of layer 1 and required as input to vertical processing of the same layer are necessary.

Practical system limitations and requirements encountered by the designer include memory size and bandwidth for the storage of the temporary data and the efficient use of both on- and off-chip storage [1]–[7]. Therefore, redesigning the data processing scheduling and the memory storage scheme allows a joint optimization of the algorithmic and architectural features according to specific system requirements. The optimum choice of these factors can be achieved by analyzing different strategies. Each of these strategies corresponds to an implementation characterized in parametric form in terms of generic architectural features such as on-chip memory size,

Manuscript received August 2001; revised April 1, 2002.

M. Ravasi and M. Mattavelli are with the Integrated Systems Laboratory (LSI) of the Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland (e-mail: massimo.ravasi@epfl.ch).

L. Tenze is with the Image Processing Laboratory (IPL), D.E.E.I. University of Trieste, I-34100 Trieste, Italy (e-mail: tenze@ipl.univ.trieste.it).

Publisher Item Identifier 10.1109/TCSVT.2002.800863.

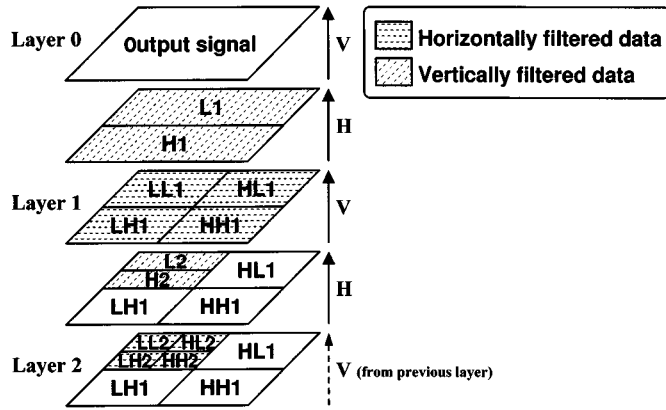


Fig. 2. 2-D DWT with Mallat-tree decomposition. The size of intermediate layers decreases twice as fast as the 1-D case. The amount of data to be filtered tends asymptotically to  $4/3$  of the size of the input signal. Since data must be filtered both horizontally and vertically, the total amount of filtered samples tends to  $8/3$  of the size of the input signal.

on-chip data-path bandwidths, overall filter complexity, external memory size, and external data-path bandwidths [1]–[3]. This paper presents the implementation of an hardware DWT decoder based on “sliding windows layer-by-layer” (SW-LbL) architecture [1], [2]. Among the different approaches studied in literature, the SW-LbL approach [1], [2] offers a good tradeoff between system performance and complexity. The main result is a sensible reduction of the bandwidth and of the size of external memory, at the price of a small increase of implementation complexity.

This paper is organized as follows. Section II presents the SW-LbL architecture [1], [2] chosen to implement the DWT decoder, comparing it with the classical architecture. Section III describes the VHDL implementation of the decoder. Section IV reports the results of the synthesis in terms of performance and system requirements. Section V concludes the paper, summarizing the main results achieved.

## II. SW-LbL DWT DECODER

### A. Classical Approach

The classical approach to 2-D DWT decoding (see Fig. 2) is to process each layer in the tree decomposition separately and to process the vertical and horizontal layers successively one after the other. The performance of this approach is strongly limited by the management of temporary data required between two successive layers and between horizontal and vertical filtering.

Let us consider an input image with resolution of  $W \cdot H$  samples encoded on  $L$  layers with Mallat-tree decomposition. As shown in Fig. 2, while processing layer 1, the amount of temporary data between horizontal and vertical filtering is equal to the size of the image to be decoded. Even with relatively small image resolutions, the memory required to store these temporary data might be, in general, too large to be implemented on-chip. Therefore, an external memory (EM) must be used and its size, measured in samples, is

$$EM_{s,\text{classical}} = WH. \quad (1)$$

The amount of data to be filtered on each layer decreases by a factor of four from one layer to the next, and the total amount of processed data along the whole tree reconstruction process is given by

$$D = \sum_{l=1}^L \frac{WH}{4^{l-1}} = \frac{4^L - 1}{3 \cdot 4^{L-1}} WH \approx \frac{4}{3} WH. \quad (2)$$

The last term of (2) approximates in excess the exact result with an error smaller than 0.4% already for  $L \geq 4$ ; hence, the approximated value for  $D$  will be used in the following expressions. Because of horizontal and vertical filtering, these data must be read and written twice on each layer, and thus

$$D_{r,\text{classical}} = D_{w,\text{classical}} = 2D = \frac{8}{3} WH. \quad (3)$$

Assuming that the compressed image bitstream is read by the decoder from the outside and that the decoded image is sent outside as the decoder output signal, the external memory is used only to store temporary data and the total amount of samples exchanged with the external memory is

$$\begin{aligned} EM_{r,\text{classical}} &= D_{r,\text{classical}} - WH \\ EM_{w,\text{classical}} &= D_{w,\text{classical}} - WH \end{aligned} \quad (4)$$

which is equivalent to

$$EM_{r,\text{classical}} = EM_{w,\text{classical}} = \frac{5}{3} WH. \quad (5)$$

The implementation of the decoder consists mainly of two 1-D linear filters with a small support interval (e.g., DWT kernels suggested in JPEG 2000 are not longer than 18 samples). These filters can be used both for vertical and horizontal processing because these two operations never occur at the same time. Thus, the implementation of a decoder according to the classical approach is rather simple, and its main disadvantage is the need of a large external memory and, above all, of a large data exchange with it. Indeed, external memory is less efficient than on-chip cache memory both in terms of available bandwidth, power consumption, and overall system complexity. The management of temporary data is thus the main bottleneck of the classical approach.

### B. SW-LbL Approach

A redesigning of the DWT algorithm, in order to change the operation scheduling, allows us to reduce the temporary data lifetime and to optimize the overall system performance [1]–[3]. The SW-LbL approach [1], [2] allows to significantly reduce both external memory size and bandwidth at the price of a slightly more complex implementation and the need of a small on-chip memory. The main idea behind this approach is to exploit data dependencies between horizontal and vertical processing in order to try to use temporary samples as soon as they are available.

The scheme of Fig. 3 shows how to manage temporary samples between horizontal and vertical filtering in order to reduce their lifetime. Let us suppose that the horizontal filters are applied first. Horizontal filters produce samples along rows, while vertical filters need input samples along columns. To generate

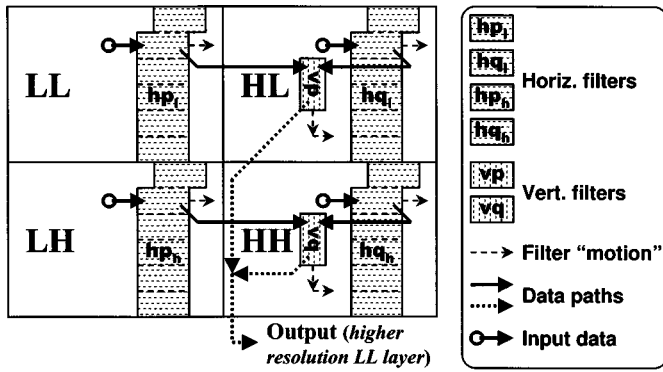


Fig. 3. SW-LbL DWT decoder. The lifetime of temporary data between horizontal and vertical filtering is minimized by using them as soon as possible after their creation. The two columns of coupled horizontal filters are virtually implemented by means of a cache memory and of one couple only of real filters working line by line.

these columns of samples with horizontal filters, a set of horizontal filters could be used, with a couple of  $p - q$  filters for each line. In this way, scheduling the horizontal filters line by line, columns of temporary data capable of feeding vertical filters are generated. Actually, it is not necessary to implement a couple of filters for each line because they never work in parallel; only one line is active at a time. It is just needed to store two columns of samples necessary as input by all the virtual horizontal filters and read them line by line to load them in parallel in a true real couple of horizontal  $p - q$  filters. These two columns behave like two windows sliding over the input image, because they cache successive columns of input samples. This is the origin of the name for this approach. In order to put the sliding windows (SW) memory on chip, to reconstruct layer  $n$  it is necessary to exchange with external memory only input layer data (layer  $n - 1$  coded signals) and output layer data (reconstructed layer  $n$ ), while all the temporary samples between horizontal and vertical filtering are managed by on-chip memory.

In general, the memory required to store temporary data between two successive layers is still too large to be implemented on-chip. Looking at Fig. 2 and keeping in mind that temporary data between horizontal and vertical filtering no longer need to be stored on external memory, it can be noticed that the maximum amount of temporary data to be managed is given by  $LL1$  signal storage that can be expressed as

$$EM_{s,SW-LbL} = \frac{WH}{4}. \quad (6)$$

Concerning data exchange with external memory, it can be observed that with respect to the classical approach it is no longer needed to read and write temporary data between horizontal and vertical filtering. That is, in (5), instead of considering  $2D$  samples [because of (3)], only  $D$  samples must be considered, as follows:

$$\begin{aligned} EM_{r,SW-LbL} &= EM_{w,SW-LbL} \\ &= EM_{r,classical} - D \\ &= \frac{5}{3}WH - D. \end{aligned} \quad (7)$$

That, according to (2), results in

$$EM_{r,SW-LbL} = EM_{w,SW-LbL} = \frac{1}{3}WH. \quad (8)$$

Considering Fig. 3, it can be observed that the size of the memory required to store the two columns of samples  $hp$  and  $hq$  depends on the size of the reconstruction filters. Such size depends on the chosen DWT kernel and on the height of the highest layer to reconstruct, that is, on the height  $H$  of the image. Let  $K_s$  be the kernel size

$$K_s = |p| + |q| \quad (9)$$

and let  $K_{s,MAX}$  be the maximum supported kernel size. This results in

$$K_{s,MAX} = |p|_{MAX} + |q|_{MAX}. \quad (10)$$

According to the DWT reconstruction process, input data must be upsampled before being processed by reconstruction filters  $p$  and  $q$ . Therefore,  $p$  and  $q$  filters may be replaced by two couples of filters  $p_e - q_e$  and  $p_o - q_o$ , whose length is half of that of the original filters, working alternately on the same input data to produce respectively output samples in even and odd positions. Therefore, the total size of the interlayer temporary memory is given by

$$IM_{s,SW-LbL} = \frac{K_{s,MAX}H}{2} \quad (11)$$

and considering that typical kernel lengths range up to 20 samples [8], this memory, referred to here as *internal memory* (IM), can be conveniently and efficiently implemented on-chip.

Every line of the SW memory holds the input data of the corresponding horizontal virtual filter, thus emulating the functionalities of a FIFO. For each  $L - H$  couple of input samples, two output samples are produced; this implies that for each output sample, a new input sample has to be stored in virtual filter's FIFO, according to expression

$$IM_{w,SW-LbL} = \sum_{l=1}^L \frac{WH}{4^{l-1}} \approx \frac{4}{3}WH. \quad (12)$$

Having to work along columns, whole lines of data in SW memory have to be reloaded, line by line, into the horizontal filter to produce each column of output data. Consequently, the amount of data read from SW memory is larger than the amount of written data

$$IM_{r,SW-LbL} = \sum_{l=1}^L \frac{WH}{4^{l-1}} \cdot \frac{K_s}{2} \approx \frac{2}{3}WH \cdot K_s. \quad (13)$$

This relatively high bandwidth requirement can be easily satisfied by the high-speed performances provided by available on-chip memory cores.

By means of a small on-chip cache, the SW-LbL approach achieves a reduction of the external memory size to a quarter of that of the classical approach, and the memory bandwidth to a fifth. In the following section, the most significant issues about

TABLE I  
HOR\_FIL

| Description                            | Value | Unit            |
|--|-------|-----------------|
| Ports                                  | 101   | #               |
| Multipliers (16x16 bits)               | 20    | #               |
| Adders (32 bits)                       | 4     | #               |
| Data registers (16 bits)               | 8     | #               |
| Kernel coefficient registers (16 bits) | 20    | #               |
| Combinatorial area                     | 2.54  | mm <sup>2</sup> |
| Noncombinatorial area                  | 0.25  | mm <sup>2</sup> |
| Net Interconnect area                  | 0.42  | mm <sup>2</sup> |
| Total area                             | 3.20  | mm <sup>2</sup> |

TABLE II  
VER\_FIL

| Description                            | Value | Unit            |
|--|-------|-----------------|
| Number of ports                        | 102   | #               |
| Multipliers (16x16 bits)               | 20    | #               |
| Adders (32 bits)                       | 4     | #               |
| Data registers (16 bits)               | 8     | #               |
| Kernel coefficient registers (16 bits) | 20    | #               |
| Combinatorial area                     | 2.52  | mm <sup>2</sup> |
| Noncombinatorial area                  | 0.26  | mm <sup>2</sup> |
| Net Interconnect area                  | 0.38  | mm <sup>2</sup> |
| Total area                             | 3.17  | mm <sup>2</sup> |

TABLE III  
FIR\_CONTROL

| Description           | Value | Unit            |
|-----------------------|-------|-----------------|
| Ports                 | 100   | #               |
| Combinatorial area    | 0.93  | mm <sup>2</sup> |
| Noncombinatorial area | 0.14  | mm <sup>2</sup> |
| Net Interconnect area | 0.18  | mm <sup>2</sup> |
| Total area            | 1.25  | mm <sup>2</sup> |

TABLE IV  
SCHEDULE

| Description           | Value | Unit            |
|-----------------------|-------|-----------------|
| Ports                 | 84    | #               |
| Combinatorial area    | 0.26  | mm <sup>2</sup> |
| Noncombinatorial area | 0.16  | mm <sup>2</sup> |
| Net Interconnect area | 0.08  | mm <sup>2</sup> |
| Total area            | 0.49  | mm <sup>2</sup> |

the hardware implementation of the SW-LbL decoder are discussed. Moreover, the main results achieved and performances related to system requirements are presented.

### III. SW-LbL DECODER IMPLEMENTATION

The architecture of a SW-LbL decoder is composed by four main blocks (see Tables I–IV): the horizontal filters block (HOR\_FIL), the vertical filters block (VER\_FIL), the state machine that synchronizes all the operations at layer level (FIR\_CONTROL), and the state machine ruling the whole system functionalities at high level, from parameters set up to decoding along all layers (SCHEDULE). It will be shown that

the most complex part of this implementation is the FIR\_CONTROL, because the synchronization of all the operations (mainly the synchronization of vertical and horizontal filtering, in steady state as well as at the borders) is not a simple task.

At the synthesis stage, the SW-LbL decoder can be scaled in terms of maximum image sizes, maximum kernel length, sample's word size, and kernel coefficient's word size. At run time, the decoder can be programmed to process any image size within the maximum image size, with any kernel of any length within the maximum kernel length.

While it has been shown that DWT kernel filters can be efficiently implemented using the lifting scheme [8], [9], it is chosen here to implement the filters with the classical filters scheme. Despite the fact that the classical filter scheme needs a larger SW memory than the lifting scheme (the total number of taps of lifting-scheme's filters is about half of that of the corresponding classical scheme), the classical scheme presents several advantages that make it preferable for several reasons.

- 1) Both in the classical scheme and in the lifting scheme, the filters are typically noncausal. Having to implement the whole kernel system as a causal system, the lifting scheme presents some obvious problems because not only must each filter be "delayed" to make it become causal, but an extra delay must also be introduced on the line to which the filter's output is added in order to preserve the correct synchronization. This extra delay can either be implemented reusing the FIFO line of the previous step's filter, when available and with enough taps, or introducing another *ad-hoc* FIFO line. In the classical scheme, both filters just need to be delayed of the same number of samples, which needs no extra delays and does not give any further synchronization problem. The same does not apply for the lifting scheme.
- 2) The programmability of the kernel is drastically more complex in the lifting scheme. In the classical scheme, it is just needed to store the kernel's coefficients, store the divisor at the end of each filter, and synchronize the output in order to take into account the delay introduced to make filters causal. In the lifting scheme, such operations must be performed at each step; in addition, it is necessary to synchronize any step in order to respect the delay introduced by the previous step. As explained earlier, depending on the kernel, either the FIFO line of the previous step's filter can be reused or another FIFO line can be used, implying the need for more multiplexers and making the overall setup by far more complex than the classical filter scheme.
- 3) According to JPEG-2000 specifications, samples at signal borders must be mirrored. Depending on the filter's impulse response, samples may be mirrored with respect either to first (last) sample or to half sampling interval before (after) first (last) sample. While with the classical scheme we can easily reuse the taps of the filters to implement this feature, with the lifting scheme this implementation is not straightforward.
- 4) As shown with (12), for each processed sample, it is necessary to write one sample only in the internal memory.

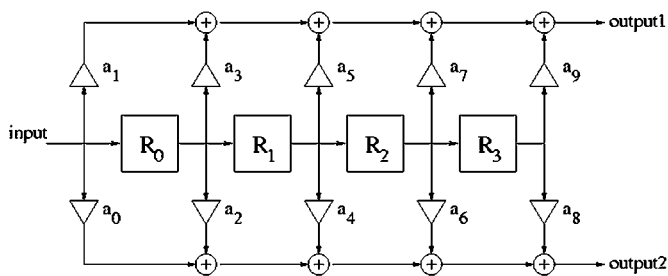


Fig. 4. Poly-phase implementation of kernel filters.

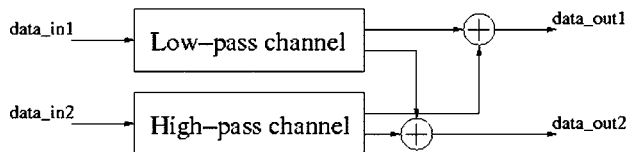


Fig. 5. Output data in even and odd positions (respectively data\_out1 and data\_out2) according to poly-phase implementation of kernel's low- and high-pass filters.

With the lifting scheme, one sample has to be written for any step with an impulse response of more than one sample, as well as for all the extra delay lines. Besides increasing the bandwidth to write on internal memory, the complexity of the management of the data to be written would increase as well.

All the filtering operations are based on integer arithmetic. Experimental results with floating-point kernels of JPEG-2000 showed that the PSNR loss of decoded image can be kept smaller than 2 dB by storing processed samples on 15-bit integers and scaling floating-point coefficients in order to map them on 9-bit integers.

In order to manage the different timing requirements among the inputs of the filters and their outputs, and among the speed of the internal memory and the external one, it has been chosen to use two clocks: the first clock (CLK) controls the output of the filters and the load and store operations of the external memory, whereas the second clock (CLK'), which is 20 times faster than CLK, controls the input of the horizontal filter and the load and store of the internal cache.

The main blocks of the decoder are the horizontal and the vertical filtering blocks: every block implements the low- and high-pass filter for DWT reconstruction. The filters have been implemented using a poly-phase approach as shown in Fig. 4.

For every input sample, two results are obtained. These results correspond to the two steps of a classical filtering process using the original data interlaced with zeros. The two outputs of both filters have to be appropriately added, as shown in Fig. 5, in order to produce samples in even and odd positions.

The input samples for the horizontal filters are stored in the internal cache while a whole line of samples has to be loaded at every CLK clock cycle to let both the horizontal filters produce an output value. Samples are loaded from internal cache at CLK' rate, in order to load all required samples within the available time according to CLK. The vertical filtering block is clocked just by CLK clock, to synchronize both the initialization process and the outputs.

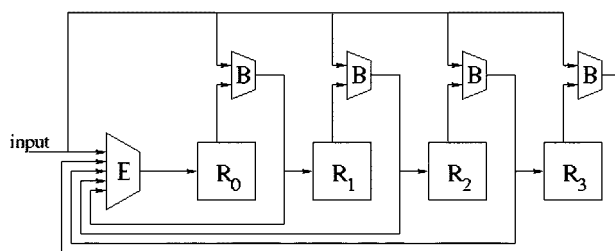


Fig. 6. Border mirroring in vertical filter block. The multiplexers "B" implement mirroring operations at the beginning of the column by allowing carrying the input sample to any register's output, while the multiplexer "E" implements the mirroring operations at the end of the column by allowing reusing as input sample the value stored in any register.

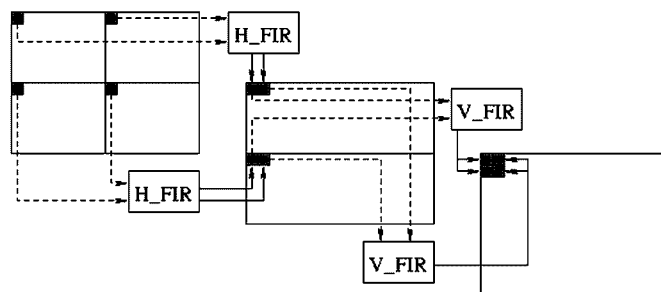


Fig. 7. Basic data flow in the decoding process. Horizontal filters (H\_FIR) produce at the same time two columns of (temporary) output data while these two columns have to be fed one after the other into vertical filters (V\_FIR).

Both horizontal and vertical borders of the input signal must be mirrored in the same way but, because of the different implementations of horizontal and vertical filters, the mirroring operations are implemented in two completely different ways. For horizontal filtering, the border mirroring is implemented reading data twice from internal cache, first backward then forward for the beginning border, and first forward then backward for the ending border. The number of memory accesses to the internal cache is not affected by mirroring because the number of written and read samples is exactly the same like in the steady state. Within the vertical filtering block, a set of multiplexers is in charge of carrying the input sample to the output of any register of the FIFO line (beginning border, multiplexers "B" in Fig. 6) or to reuse samples already in the FIFO line itself as input value (ending border, multiplexer "E" in Fig. 6). In both horizontal and vertical blocks, all the mirroring operations depend on parameters to be set during system initialization in order to correctly mirror input data according to the chosen kernel and image sizes.

The most critical issue during the decoding process is the synchronization of operations between HOR\_FIL and VER\_FIL. To describe how these two blocks interact, it is necessary to distinguish between columns in even and odd position. In order to produce two decoded samples along a column, the filter blocks are fed with four samples, as shown in Fig. 7. HOR\_FIL block produces two samples at the same time, the first in an even column and the second on the same row in the following odd column. Since the vertical block has to process one column at a time, the first sample produced by horizontal block can be passed immediately to vertical block, while the other sample has to be stored in the internal cache in order to delay its use

after current column has been completely processed. Seemingly, VER\_FIL needs two consecutive input samples on a column before being able to compute two output samples. Therefore, it has to wait for HOR\_FIL to process a “square” of four samples on two rows before producing two output samples.

While producing an even column of output samples, HOR\_FIL and VER\_FIL work together. Each clock cycle HOR\_FIL produces two temporary samples on even and odd columns; the two on the even column are immediately processed by VER\_FIL. Producing the successive odd column, only VER\_FIL works, using for input the samples previously produced by HOR\_FIL and stored in the internal cache. In both cases, VER\_FIL produces two samples at the same time, each two clock cycles. The first sample is passed immediately to the output, while the second one is delayed by one clock cycle in order to produce, in steady state, one output sample at each clock cycle.

The FIR\_CONTROL block is basically a state machine in charge of synchronizing all the steps to decode one layer. First it resets the filtering blocks, initializing them to process a new layer, then it controls the behavior of HOR\_FIL and VER\_FIL in order to synchronize the two blocks to decode a layer, take care of border mirroring, take into account the delay introduced by causal kernel filters, and manage the accesses to internal cache.

The SCHEDULE block controls the whole decoding process interacting with FIR\_CONTROL. Its main tasks are initializing FIR\_CONTROL to start decoding a layer, controlling input and output operations, and managing the transition from one layer to the next when FIR\_CONTROL notifies that a layer has been completely processed.

#### IV. PERFORMANCE AND REQUIREMENTS

This section presents the results obtained with a VHDL implementation of the SW-LbL DWT decoder. The decoder has been synthesized and validated in 0.35- $\mu\text{m}$  CMOS technology.

The parameters chosen to customize the implementation of the decoder and the tables of the corresponding results obtained after synthesis are:

- maximum image size:  $768 \times 512$  pixels;
- maximum kernel length: ten taps for both high- and low-pass channel;
- data samples' word-size: 16 bits;
- kernel coefficients' word-size: 16 bits.

We tested the synthesized chip using JPEG2000's  $7 \times 9$  floating-point kernel, with an image coded over six layers. The steady-state throughput of the decoder is one sample per clock cycle and, according to (2), we find that the minimum time required to decode an image is

$$T_{\min} = D \quad [\text{CLK cycles}]. \quad (14)$$

Because of the delays introduced to make the kernel's filters causal, and because of mirror operations at layers' borders, the total simulated decoding time is slightly greater than the theoretical minimum  $T_{\min}$

$$T = 1.04 \cdot D \quad [\text{CLK cycles}]. \quad (15)$$

The bandwidth required between the external memory and the chip and between the horizontal filters and internal cache meet the theoretical values shown respectively in (5), (12), and (13). The SW on-chip cache memory is composed by two blocks of 2560 words of 16 bits.

#### V. CONCLUSION

This paper presents the implementation of a JPEG-2000 compliant hardware DWT decoder for Mallat-tree decomposition. The decoder is capable of image decoding applications, both with integer and floating-point kernels. The customizable design allows setting the basic features of the decoder, such as maximum image size, maximum kernel length, and computational precision. The synthesized decoder may be programmed to process images with different sizes, coded on a different numbers of layers, and using different kernels, while always guaranteeing the correct mirroring operations at layers' borders. By means of an on-chip cache and of a specifically optimized scheduling of operations, the decoder requires a minimum number of accesses to external memory, making it suitable for low-power embedded applications.

#### REFERENCES

- [1] M. Ravasi, M. Mattavelli, and D. J. Mlynek, “Scheduling strategies for 2D wavelet coding implementations,” in *Proc. X Eur. Signal Processing Conf.*, vol. II, Tampere, Finland, Sept. 2000, pp. 969–972.
- [2] M. Ravasi, M. Mattavelli, D. J. Mlynek, A. Buttar, and S. Soudagar, “Wavelet image compression for mobile/portable applications,” *IEEE Trans. Consumer Electron.*, vol. 45, pp. 794–803, Aug. 1999.
- [3] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, “Optimal memory organization for scalable texture codecs in MPEG-4,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 218–243, Mar. 1999.
- [4] C. Chakrabarti, M. Vishwanath, and R. Owens, “Architectures for wavelet transforms,” *VLSI Signal Processing VI—IEEE Special Publications*, pp. 507–515, 1993.
- [5] M. Vishwanath, “The recursive pyramid algorithm for the discrete wavelet transform,” *IEEE Trans. Signal Processing*, vol. 42, pp. 673–676, Mar. 1994.
- [6] T. C. Denk and K. K. Parhi, “Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing,” in *IEEE Int. Symp. Circuit and Systems*, vol. 3, London, U.K., May 1994, pp. 77–80.
- [7] G. Lafruit and J. Bormans, “Graceful Degradation Parameters for a Scalable Wavelet Codec,” ISO/IEC JTC1/SC29/WG11/MPEG97/M2655, Fribourg, Switzerland, 1997.
- [8] C. Chui, “Integer Wavelet Transforms,” TeraLogic, Inc., Geneva, Switzerland, ISO/IEC JTC1/SC29/WG1/N769, 1998.
- [9] W. Sweldens and P. Schröder, “Building your own wavelets at home,” in *Wavelets in Computer Graphics*, 1996, ACM SIGGRAPH Course Notes, pp. 15–87.
- [10] Y. Sheng, “Wavelet transform,” in *The Transforms and Applications Handbook*, A. D. Poularikas, Ed. Boca Raton, FL: CRC, 1996, ch. 10.



**Massimo Ravasi** was born in Lecco, Italy, in 1968. He received the degree in electrical engineering from the Politecnico di Milano, Milano, Italy, in 1997. He is currently working toward the Ph.D. degree at the Integrated Systems Laboratory (LSI) Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.

From October 1997 to March 1998, he was a part-time Collaborator at the Laboratorio S.I.A., Politecnico di Milano. In April 1998, he joined the EPFL. His research interests are system design, complexity analysis, image compression, and signal processing.



**Livio Tenze** was born in Trieste, Italy, in 1973. He received the degree in electronic engineering in October 1998 from the University of Trieste, where he is currently working toward the Ph.D. degree.

His research interests include image and video processing, multimedia applications, old motion picture restoration, image compression via JPEG 2000, and ASIC development using VHDL.



**Marco Mattavelli** received the Diploma in electrical engineering from the Politecnico di Milano, Milano, Italy, in 1987, and the Ph.D. degree from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 1996.

He joined Philips Research Laboratories, Eindhoven, The Netherlands, in 1987, where his work involved channel and source coding for optical recording electronic photography and signal processing of TV and HDTV signals. He then joined EPFL, where he is currently a Scientific

Advisor in the Integrated Systems Laboratory. His main research interests are architectures and system for video coding, the application and implementation of combinatorial optimization techniques for image analysis, and tools for the aid to architecture design of complex systems. He is also involved in ISO-MPEG standardization activities, for which he is currentl Chair of the MPEG Implementation Studies Group.