Journal of VLSI Signal Processing 2007
© 2007 Springer Science + Business Media, LLC. Manufactured in The United State.
DOI: 10.1007/s11265-007-0139-5

Implementation and Comparison of the 5/3 Lifting 2D Discrete Wavelet Transform Computation Schedules on FPGAs

MARIA E. ANGELOPOULOU AND PETER Y. K. CHEUNG
Department of Electrical and Electronic Engineering, Imperial College London, Exhibition Road, London, SW7 2BT, UK

KONSTANTINOS MASSELOS
Department of Computer Science and Technology, University of Peloponnese, Tripolis, 22100, Greece

YIANNIS ANDREOPOULOS
Department of Electronic Engineering, Queen Mary University of London, Mile End Road, London, E1 4NS, UK

Received: 14 April 2007; Revised: 23 July 2007; Accepted: 9 August 2007

Abstract. The suitability of the 2D Discrete Wavelet Transform (DWT) as a tool in image and video compression is nowadays indisputable. For the execution of the multilevel 2D DWT, several computation schedules based on different input traversal patterns have been proposed. Among these, the most commonly used in practical designs are: the row–column, the line-based and the block-based. In this work, these schedules are implemented on FPGA-based platforms for the forward 2D DWT by using a lifting-based filter-bank implementation. Our designs were realized in VHDL and optimized in terms of throughput and memory requirements, in accordance with the principles of both the schedules and the lifting decomposition. The implementations are fully parameterized with respect to the size of the input image and the number of decomposition levels. We provide detailed experimental results concerning the throughput, the area, the memory requirements and the energy dissipation, associated with every point of the parameter space. These results demonstrate that the choice of the suitable schedule is a decision that should be dependent on the given algorithmic specifications.

Keywords: discrete wavelet transform, FPGA, implementation, lifting scheme, 5/3 lifting filter-pair, row-column, line-based, block-based, comparison

1. Introduction

The two-dimensional Discrete Wavelet Transform (2D DWT) is nowadays established as a key operation in image processing. In the area of image compression, the 2D DWT has clearly prevailed against its predecessor, the 2D Discrete Cosine Transform. This is mainly because it achieves higher compression ratios, due to the subband decomposition it involves, while it eliminates the `blocking' artifacts that deprive the reconstructed image of the desired smoothness and continuity. The high algorithmic performance of the 2D DWT in image compression justifies its use as the kernel of both the JPEG-2000 still image compression standard [1] and the MPEG-4 texture coding standard [2].

1.1. The Dyadic Decomposition

The 2D DWT can be considered as a `chain' of successive levels of decomposition as depicted in Fig. 1. Because the 2D DWT is a separable transform, it can be computed by applying the 1D DWT along the rows and columns of the input image of each level during the horizontal and vertical filtering stages. Every time the 1D DWT is applied on a signal, it decomposes that signal in two sets of coefficients: a low-frequency and a high-frequency set. The low-frequency set is an approximation of the input signal at a coarser resolution, while the high-frequency set includes the details that will be used at a later stage during the reconstruction phase.

This procedure, presented in Fig. 1, is known as the dyadic decomposition of the image, and its impact upon the image's pixels can be presented by the diagram of Fig. 2, for the case of three decomposition levels. The shaded areas in Fig. 2 represent the low-frequency coefficients that comprise the coarse image at the input of each level.

Let us briefly describe the steps of this decomposition. The input of level j is the low-frequency 2D subband LL_j, which is actually the coarse image at the resolution of that level. In the first level, the image itself constitutes the LL image block (LL_0). The coefficients L (H), produced after the horizontal filtering at a given level, are vertically filtered to produce subbands LL and LH (HL and HH). The LL subband will either be the input of the horizontal filtering stage of the next level, if there is one, or will be stored, if the current level is also the last one. All LH, HL and HH subbands are stored, to contribute later in the reconstruction of the original image from the LL subband.

1.2. The 1D DWT

In Fig. 1, the units that implement the 1D DWT are depicted as black boxes. These are now considered in detail. Two main options exist for the implementation of 1D DWT: the traditional convolution-based implementation [3] and the lifting-based implementation [4, 5].

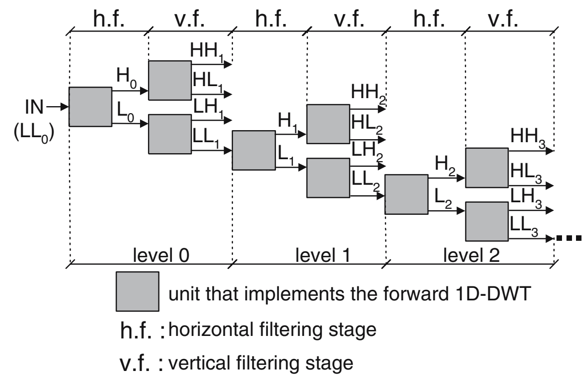
h.f.
v.f.
IN
(LL0)
H0
L0
HH1
HL1
LH1
LL1
H1
L1
HH2
HL2
LH2
LL2
H2
L2
HH3
HL3
LH3
LL3
level 0
level 1
level 2
unit that implements the forward 1D-DWT
h.f. : horizontal filtering stage
v.f. : vertical filtering stage

Figure 1. The 2D DWT decomposition as a 'chain' of successive levels.

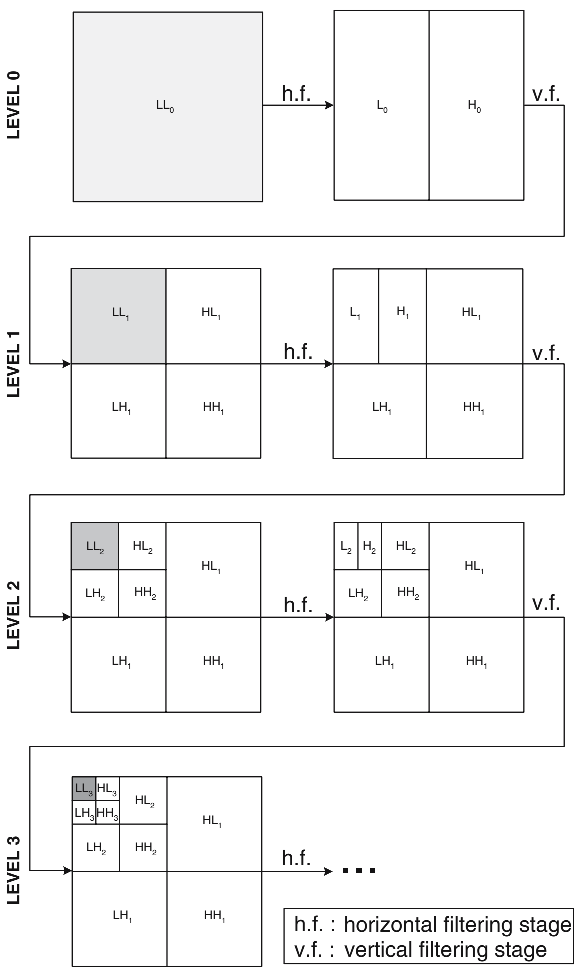

Figure 2. Diagrammatic representation of the dyadic decomposition for three decomposition levels.

The convolution-based 1D DWT The conventional convolution-based 1D DWT of [3] is presented in Fig. 3a. As shown in Fig. 3a, this consists of two analysis filters, h (low-pass) and g (high-pass), followed by subsampling units. The signal $x[n]$ is decomposed into the approximation (low-frequency) signal $lp[n]$ and the detail (high-frequency) signal $hp[n]$. Note that in the structure of Fig. 3a the downsampling is performed after the filtering has been completed. This is clearly inefficient since, in this case, half of the calculated coefficients are redundant, and the filtering is realized at full sampling rate.

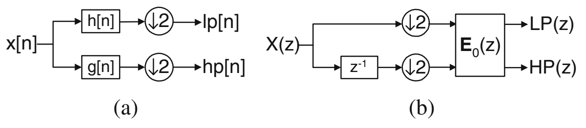

Figure. 3. The convolution-based implementation of the forward 1D DWT. The conventional filtering-and-downsampling structure (**a**). Using the polyphase matrix of the analysis filter-bank (**b**).

Early research on filter-bank design proved that the execution of 1D DWT can be accelerated by using the polyphase matrix of the filter-bank, instead of the conventional filtering-and-downsampling structure of Fig. 3a. As Fig. 3b shows, the signal is split into two signals (polyphase components) at half of the original sampling rate. The downsampling is now performed prior to the actual filtering, thereby avoiding the calculation of coefficients that will later be discarded. The polyphase components of the signal are filtered in parallel by the corresponding filter coefficients, producing the same result as if the downsampling was performed as described in [3].

The analysis polyphase matrix for Fig. 3b is defined (in the Z-domain) as:

$$\mathbf{E}_0(z) = \begin{bmatrix} H_e(z)H_o(z) \\ G_e(z)G_o(z) \end{bmatrix} \tag{1}$$

where $H_e(z)$ and $H_o(z)$ denote the Type-I even and odd polyphase components of the corresponding low-pass analysis filter, and $G_e(z)$ and $G_o(z)$ denote the Type-I even and odd polyphase components of the corresponding high-pass analysis filter.

Using the analysis polyphase matrix of (1), the wavelet decomposition can be written (in the Z-domain) as:

$$\begin{bmatrix} LP(z) \\ HP(z) \end{bmatrix} = \mathbf{E}_0(z) \begin{bmatrix} X_e(z) \\ X_o(z) \end{bmatrix}, \tag{2}$$

where $LP(z)$ denotes the approximation at the coarser resolution, $HP(z)$ denotes the detail signal, and $X_e(z)$ and $X_o(z)$ denote the Type-I even and odd polyphase components of the signal $X(z)$.

The convolution-based 1-D DWT suffers from high computational complexity and high memory utilization requirements.

The lifting-based 1D DWT The lifting scheme reduces the transform computational requirements by factorizing the polyphase matrix of the DWT into elementary matrices.

We will briefly discuss the principles of the forward lifting. More information on the lifting scheme can be found in [4, 5]. As it is proven in [4], if (H, G) is a perfect-reconstruction filter-pair, the following factorization of matrix $\mathbf{E_0}(\mathbf{z})$ of (1), into lifting steps, is feasible:

$$\mathbf{E}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & U_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -P_i(z) & 1 \end{bmatrix} \tag{3}$$

where K is a constant and m is the number of predict-and-update steps. They both depend on the type of filter-pair that is used.

Note that the lifting factorization of (3) is not unique. That is, for a given wavelet transform we can often find multiple ways of factorizing its polyphase matrix.

The numerical factorization of (3) is represented by the schematic diagram of Fig. 4. The forward lifting scheme consists of the following steps:

1. The splitting step, where the signal is separated into even and odd samples.
2. The prediction steps, associated with the predict operator $P_i(z)$.
3. The update steps, associated with the update operator $U_i(z)$.
4. The scaling step, indicated by the scaling factors $1/K$ and K.

The inverse transform is realized by traversing the schematic of Fig. 3b from right to left (reverse signal flow) and switching the signs of the predict and update operators as well as the scaling factors. Because of the complete reversibility of the lifting scheme even if non-linear predict and update operators are used in the schematic of Fig. 3b, we can use a rounding operator to ensure integer form for all the data produced, throughout the execution of the 2D DWT. In this case, the 2D DWT is completely reversible, and, therefore, lossless. Such transforms are known as integer-to-integer transforms and are extremely useful in lossless coding. To build an integer version of a given wavelet transform, if scaling is present, then the scaling step should be either split further [6, 7] or omitted. In fact, the scaling performed at the end of each decomposition level in the conventional decomposition can be

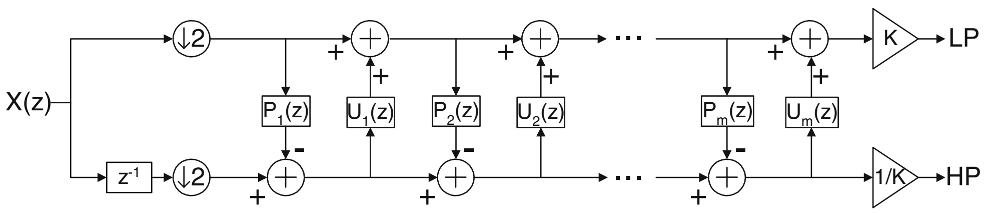

Figure 4. The lifting-based implementation of the forward 1D DWT.

skipped altogether [8], or incorporated into the subsequent encoding or processing stage of each bitplane and as a result it is not explicitly considered in this paper.

Due to the lower computational cost, the flexibility offered in the transform factorization, and the perfect reconstruction property, the lifting-based implementation is generally preferable for the design of optimized systems.

1.3. The 2D DWT

Several computation schedules have been proposed, to implement the 2D DWT. In practical designs, the most commonly used computation schedules are: the row–column (RC) [3], the line-based (LB) [9] and the block-based (BB) [10]. The simplest of these is RC, which adopts the level-by-level logic of Fig. 1. However, such an approach necessitates the use of large off-chip memory blocks as the only source of the filter's inputs. Contrary to RC, both LB and BB involve an on-chip memory structure that operates as a cache for the original image, minimizing the accesses of the large memory blocks. Thus, memory utilization and memory-access locality are improved. The main difference among LB and BB concerns the way the original image is traversed. Specifically, in LB, non-overlapping groups of lines are processed, whereas, BB operates using non-overlapping blocks of the image.

Related work Implementations of 2D DWT computation schedules can be found in [11] and [12]. In [13] a combined lifting line-based FPGA implementation for the single-level 5/3 and 9/7 2D DWT is presented.

In [14, 15] and [16], 2D DWT computation schedules have been compared on a theoretical basis. In [17] and [18], they are compared on programmable architectures and on a VLIW DSP, respectively. Even though the above comparisons are particularly enlightening, none of them is based upon hardware implementations. Thus, the implementations involved do not take advantage of the implementation efficiency and the parallelism in data processing that hardware could offer. In addition, the vast majority of comparisons of the different alternatives focuses on convolution-based realizations and lifting is not considered. A discussion of VLSI architectures which also considers lifting-based DWT modules is presented in [19]. However, very few performance figures are presented for the multilevel lifting 2D DWT, which is the case of interest in image coding applications. Hence this important case remains largely unexplored so far in the relevant literature. Our initial results reported in [20] present some initial performance figures, however in this paper we extend these results and present energy performance figures for the transform realization.

Contribution of this paper In this paper, the three major 2D DWT lifting-based computation schedules are implemented on FPGA-based platforms and compared in terms of performance, area and energy requirements. The computation schedules are compared for different image sizes (M×M) and number of levels (L) of the transform. To the best of our knowledge no detailed comparisons of hardware implementations of the three major lifting-based 2D DWT computation schedules exist in the literature. This comparison will give significant insight on which schedule is most suitable for given values of the relevant algorithmic parameters.

Structure of this paper Section 2 presents decisions we made that are, for comparison reasons, common in all our FPGA implementations. Sections 3.1, 3.2 and 3.3 describe how we implemented RC, LB and BB, respectively. In Section 4, we discuss the results of the three FPGA implementations, concerning the throughput, the number of FPGA slices, and the memory and energy requirements. In Section 5 we briefly discuss what type of optimizations would be more appropriate for each schedule. Finally, Section 6 offers our conclusions.

2. Common Implementation Decisions and Assumptions

The comparison of the main 2D DWT computation schedules is performed on the basis of some common implementation decisions and assumptions. These are concerned with the memory architecture used to store the image (image memory) and the filtering structure used to compute the DWT.

2.1. Image Memory

In this comparison, a single-port image memory is considered. The image memory is usually off-chip. However, it may also be on-chip, if we are dealing with small frames and/or using large FPGA devices. Considering the fact that the image memory can sometimes be on-chip, we made our comparison as generic as possible by using a common clock for the image memory and the rest of the system.

Employing a single-port image memory in our analysis adds flexibility as far as the available memory is concerned, as many boards do not include multi-port large memory blocks. Also, since the image memory is usually off-chip, the fact that single-port off-chip RAMs consume less energy per access than multi-port ones is definitely something to consider [14, 19]. In data-intensive algorithms, such as the 2D DWT, memory accesses are highly frequent. Thus, when the image memory is off-chip, from an energy perspective single-port RAMs are ideal, trading off the higher performance of multi-port RAMs [19].

Traditionally, two memory blocks are used in image processing systems: one to store the original image, and one for the outputs. To avoid the second block, we used the in-place mapping scheme: the filter's outputs are written over memory contents that are already consumed and no longer needed. To adopt this scheme, each memory location should have the same bit-width as the outputs of the transform.

The in-place mapping scheme is illustrated in Fig. 5. The dyadic decomposition is applied on a hypothetical 8×8 original image and the outputs of each stage are written in memory locations that have already been read and filtered. The shaded areas, in Fig. 5, represent the pixels of the coarse image at the input of each level. It might be interesting for the reader to compare Fig. 5 with Fig. 2, to see how the pixels of the different 2D subbands, of the dyadic decomposition diagram, are distributed over the memory array. Note that in Fig. 5 every distinct square represents one pixel and bares the name of the 2D subband in which it belongs. The decomposition of the hypothetical 8×8 original image cannot go beyond level 2, as there are not enough pixels in the LL_2 subband to proceed with the filtering operations.

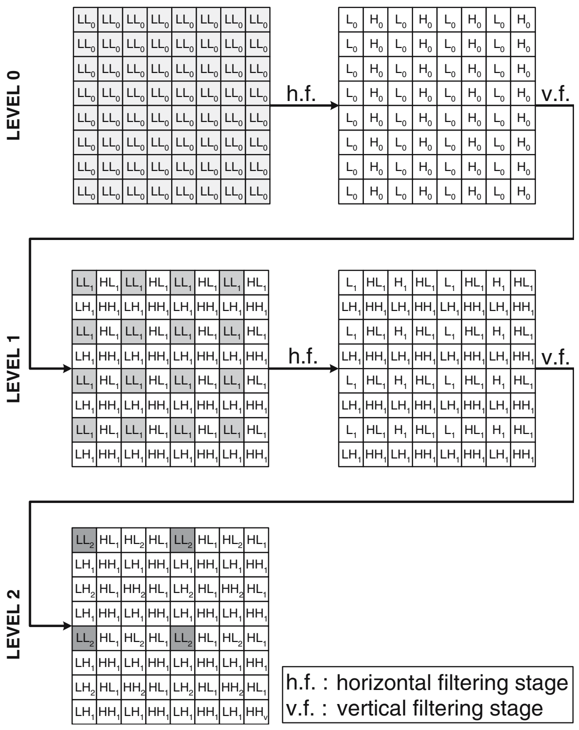
LEVEL 0
LEVEL 1
LEVEL 2
h.f.
v.f.
h.f. : horizontal filtering stage
v.f. : vertical filtering stage

Figure 5. The in-place mapping scheme. The dyadic decomposition is applied on a hypothetical 8×8 original image.

2.2. Filter Implementation

A single filter is used in all three implementations, introducing the minimum hardware cost. That is, the same single filter is shared among all levels and also among the vertical and the horizontal filtering stages within each level. The choice of using a single filter is mostly appropriate for an RC architecture that uses a single-port RAM.

The 5/3 lifting-based filter-pair As for the type of filter, considering the advantages that the lifting scheme offers, we used a lifting-based filter, instead of a traditional convolution-based filter. Two types of lifting-based filter sets are mainly used in implementations of the DWT, the 5/3 lifting filter-pair [5] and the 9/7 lifting filter-pair [4]. We used the 5/3 lifting filter-bank, as it is more hardware efficient: for the same number of pipeline stages, it has a significantly smaller critical path than the 9/7 lifting filter-bank, while occupying significantly smaller area. This is due not only to the smaller number of lifting steps required, but also to the simplicity of the multiplying units: only two simple multiplications are involved (by (1/2) and by (1/4)), that can be implemented with simple shifters.

For the conventional 5/3 filter-pair, the analysis low-pass filter h (Fig. 3) has 5 coefficients, while the analysis high-pass filter g (Fig. 3) has 3 coefficients. The filter coefficients are the following:

– Low-pass filter h: $\{-1/8, 2/8, 6/8, 2/8, -1/8\}$
– High-pass filter g: $\{-1/2, 1, -1/2\}$

The factorization of the polyphase matrix of the conventional 5/3 filter-bank renders two elementary matrices. Therefore, the lifting version of the 5/3 filter-bank will include only one predict-and-update step, or equivalently m will equal 1 (see Section 1.2). For each pair of input samples, $2n$ and $2n+1$, the lifting equations of this filter-bank are the following:

$$HP[2n+1] = X[2n+1] - \lfloor \frac{X[2n] + X[2n+2]}{2} \rfloor \tag{4}$$

$$LP[2n] = X[2n] + \lfloor \frac{HP[2n-1] + HP[2n+1] + 2}{4} \rfloor \tag{5}$$

where X are the signal samples, HP is the high-frequency output coefficient, LP is the low-frequency output coefficient, and $\lfloor . \rfloor$ represents the floor operator. The floor operator ensures an integer-to-integer lossless transform.

Hardware implementation of the 5/3 lifting filter-pair In our hardware implementation of the filter, a three-word FIFO stores inputs $X[2n+1]$, $X[2n]$, $X[2n+2]$, and a register stores $HP[2n-1]$, which was calculated in the previous lifting step. According to the above equations, for a new pair of output coefficients to be computed, two (and not one) new filter inputs are needed. Thus, a filtering operation will take place, and a pair of output coefficients will be produced, every two cycles.

Our hardware implementation of the 5/3 lifting filter-pair is shown in Fig. 7. Registers r1, r2 and r3 constitute the FIFO. Observing Fig. 8, one concludes that the filter's behavior is determined according to whether the initialization phase is over. The *initialization_signal* controls multiplexers m1 and m2, which determine if the data flows with a step of one or a step of two. Thus, the filter-bank might behave in one of the two following ways:

1. The *initialization mode*, where the data flows with a step of one. During the initialization phase, two samples are mirrored around the first sample, and a simple shifting takes place in the FIFO, as shown in Fig. 6.
2. The *normal mode*, where the data flows with a step of two. After the initialization phase has been over, the *we_NormalMode* signal will be forced to high when the FIFO should be written. Since the filtering operation should take place every two cycles, the *we_NormalMode* signal gets high every second cycle. Thus, the pattern shown in Fig. 7 is achieved. The same pattern applies for the finalization mirroring to be executed (Fig. 6), but the source of r1's input is now the FIFO itself, eliminating the need to access memory during this step.

In RC, only one input enters the filter-bank per cycle, since the single-port image memory is the only source of input coefficients. Thus, for RC, the filter-bank of Fig. 8 is ideal. On the contrary, as we will see in the following sections, both LB and BB involve

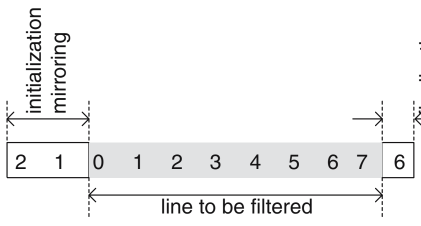

Figure 6. Mirroring at the borders of an 8-pixel incoming line, for a 5/3 lifting filter-bank.

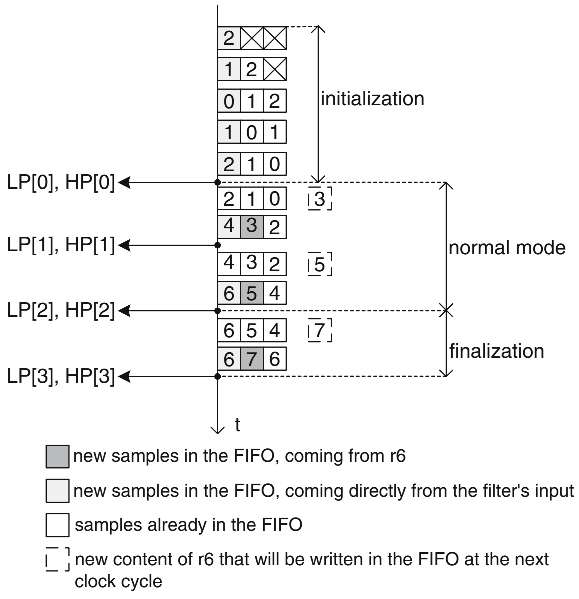
initialization
normal mode
finalization
LP[0], HP[0]
LP[1], HP[1]
LP[2], HP[2]
LP[3], HP[3]
t
new samples in the FIFO, coming from r6
new samples in the FIFO, coming directly from the filter's input
samples already in the FIFO
new content of r6 that will be written in the FIFO at the next clock cycle

Figure 7. The contents of the FIFO in respect to time for the filtering of an 8-pixel line.

multi-port on-chip buffers, that can supply the filter-bank with more than one inputs per cycle. In order to make the best possible use of the parallelism offered, we made a few small changes to the filter-bank of Fig. 8, to incorporate a multiple-input function (Fig. 9). Now, two or three inputs can be inserted in the filter-bank at a single cycle. However, during the horizontal filtering at level 0, the samples will be drawn from the single-port memory, just like in the case of RC. Thus, during the horizontal filtering at level 0, the new filter-bank behaves in a single-input mode, apart from the multiple-input mode in which it functions in any other case.

3. Implementing the Computation Schedules

In this section we will describe how we implemented the RC, LB and BB 2D DWT computation schedules. Fig. 10 presents a generic block diagram of a 2D DWT FPGA-based system that uses an off-chip image memory (which is the most common case). The dashed line indicates that on-chip buffers are not used in all of the three schedules.

3.1. Row–Column Implementation

The RC is implemented by applying the forward 1D DWT in both the horizontal and the vertical direction of the image, for a chosen number of levels, in the way shown in Fig. 2. Specifically, in any given level, in order to proceed to the vertical filtering, of the current level's *LL* image block, the horizontal filtering should be complete. In addition, in order to proceed to the next level, the filtering at the previous level should be finished. Figures 11 and 12 present the flowchart and the block-diagram of RC, as implemented. Related to the generic block-diagram of Fig. 10, in the block diagram of RC (Fig. 12) the on-chip buffering block is omitted.

The RC architecture is the one with the simplest control path. The parallelism achieved during the filtering operations depends on the number of ports of the image memory. Its major disadvantage is the lack

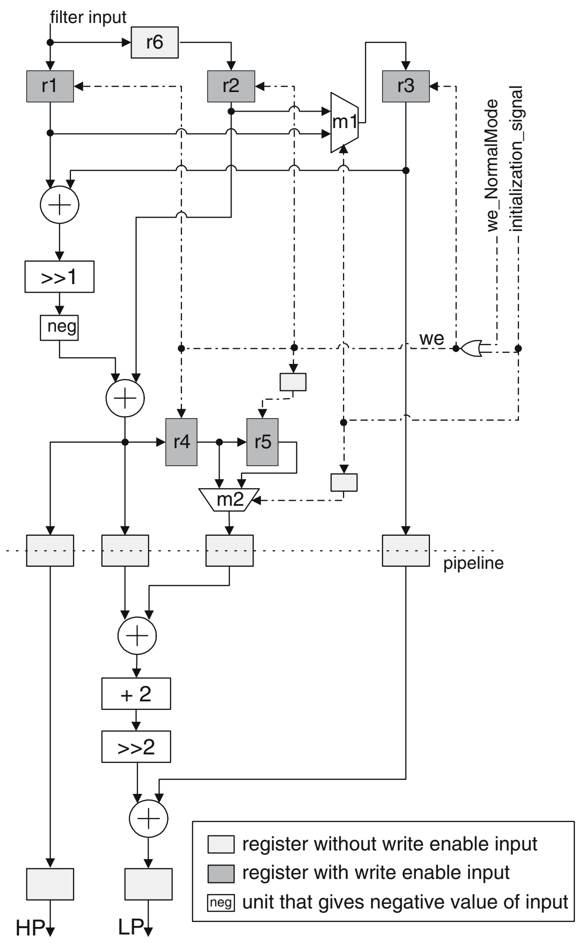
filter input
r6
r1
r2
r3
m1
we_NormalMode
initialization_signal
>>1
neg
we
r4
r5
m2
pipeline
+2
>>2
register without write enable input
register with write enable input
unit that gives negative value of input
HP
LP

Figure 8. Hardware implementation of the 5/3 lifting filter-pair, designed to perform the 1D DWT. The write enable signal (*we*) determines if the registers with write enable inputs will be written.

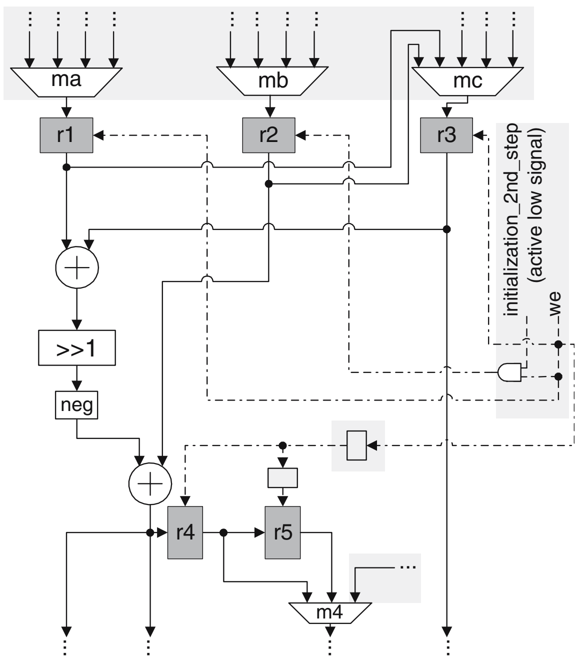

Figure 9. This filter-bank derives from that of Fig. 7, by applying a few changes (the shaded areas), to incorporate a multiple-input function. The lower part is not shown, as it remains the same.

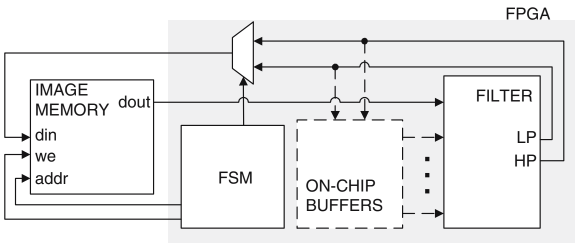

Figure 10. Generic block diagram of a system which executes the 2D DWT using an off-chip image memory.

of locality, due to the use of off-chip large memory blocks. This decreases the performance and increases the power consumption.

3.2. Line-Based Implementation

Figure 13 presents the flowchart of the low-level LB algorithm, as implemented. In Fig. 13 r denotes the current row of the original image and j denotes the current level of processing. The counter $COUNT(j)$ determines (a) if there exists sufficient information for a vertical filtering to occur at level $j+1$ and (b) which is the filtering mode of the next vertical filtering at level $j+1$. Specifically, if $COUNT(j)=4$, a vertical filtering will occur at level $j+1$ in initialization mode, whereas if $COUNT(j)=2$, it will occur in normal mode. If $COUNT(j)=3$ or $COUNT(j)=1$, there is not sufficient information for a vertical filtering to occur at the current stage, but as soon as enough coefficients are produced a vertical filtering will occur in initialization or normal mode, respectively.

Figure 14 shows the block diagram of the LB architecture. Contrary to the RC, the LB uses on-chip buffers. The on-chip buffers used in all of the levels are included in the shaded area of Fig. 14. In Fig. 15 the buffers of level j are isolated. The LB uses these on-chip buffers, to store coefficients of intermediate levels which will be used at subsequent levels. This improves the memory-access locality compared to RC, and, hence, improves the performance. Moreover, contrary to RC where the single-port image memory imposes a serial nature to the filtering operations, these buffers may be multi-port to increase parallelism.

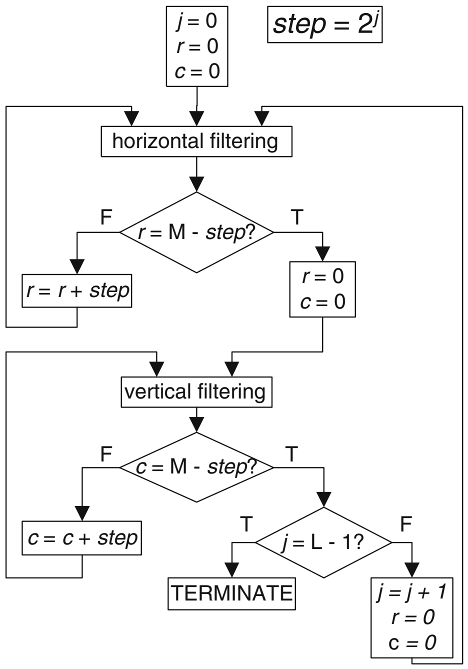

Figure 11. Flowchart of the RC algorithm as implemented (r/c= current row/column, j=current level).

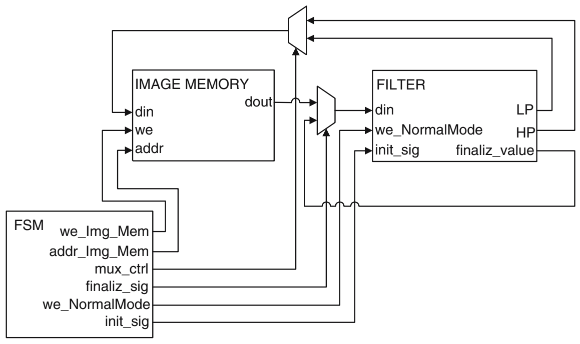

Figure 12. Block diagram of the RC architecture.

A group of lines is processed up to the final level, and a filter's output is stored in image memory only if it will be used as it is during reconstruction and never again during decomposition. Thus, LH, HL and HH coefficients are stored after the vertical filtering at any level, whereas the LL coefficients are stored only at the last level (Fig. 13).

After the completion of a vertical filtering at level $j-1$, the resulting LL_j coefficients are written in R(j). Buffer R(j) will then be horizontally filtered and the resulting coefficients will be written, depending on the current stage of level j's vertical filtering, in one of the following: C(j), R(j) (implementing the in-place mapping scheme) or buf1(j) (only during the initialization phase of the vertical filtering at level j, when buf1(j) is still empty).

Using the still empty buf1(j) during vertical initialization, we eliminate the extra line buffer that would store the extra information needed for the initialization mirroring to occur. When initialization is over, that is during the normal-mode vertical filtering, the value of the FIFO's first register (r1) and the high-frequency output of the current lifting step, are stored in buf1(j) and buf2(j), respectively. Thus, the coefficients of the preceding lifting step are retrieved at the current step from buf1(j) and buf2(j). In this way, a continuity in the vertical filtering of each column is created, since in LB the vertical filtering, contrary to the horizontal filtering, is *not* inherently continuous. During vertical finalization, the values of buf1(j) are also the samples that are mirrored.

The filter used in LB, is that of Fig. 9, which incorporates a multiple-input mode to take advantage of the multi-port nature of the on-chip RAMs. Figures 16 and 17 depict how the initialization and normal mode vertical filtering of each column is executed, and also show the multi-port filter's behavior.

3.3. *Block-Based Implementation*

The BB is implemented bringing on chip blocks of the original image. Traditionally, the size of these blocks is equal to $2^L \times 2^L$, to allow the generation of either an LL_L/LH_L or an HL_L/HH_L pair, L denoting the final level. Thus, an on-chip memory of equal size should be used, where blocks are temporally stored. This memory is known as Inter-Pass Memory (IPM) [10]. The RC algorithm is then applied on the block, up to the last level, the decomposition of the block is written back to image memory, and the next block is brought on chip. The traditional version of BB demands complicated control and addressing [14], is not effective in streaming applications and imposes high memory requirements (for six levels of decomposition the size of the IPM would be 4096 words).

The BB version that we implemented is the one that requires the minimum size of local buffers and, also, simplifies as much as possible the control and the addressing. Each level has its own IPM, where coefficients LL are stored to be filtered horizontally. The size of IPM(j) is such that allows the generation of an L_j/H_j pair at level j. As we have seen, for a new L_j/H_j pair to be generated, two new input coefficients should enter the filter-bank. Thus, the size of IPM(j), where $j = 1, 2, \ldots, L-1$, will be only 2 words. No IPM is needed for level 0, as the filter's inputs come straight from image memory.

In the previous section, we saw that the vertical filtering is not inherently continuous in LB. In order for the vertical filtering to be executed correctly, a group of line-buffers was used. In BB, the horizontal—and not only the vertical—filtering is also deprived of inherent continuity. Therefore, apart from line buffers that will be used to create vertical continuity, in order to create continuity in the horizontal filtering, two double-word registers will now be used in each level. Specifically, IPM(j) will store the input samples, that will enter the filter's FIFO, and bufH(j) will be needed to store the two intermediate results of horizontal filtering. So, the total storage place needed to create horizontal continuity is much smaller than that needed for the vertical filtering to be continuous. This is because we traverse the input image first horizontally and then vertically.

In vertical filtering, line-buffers buf1(j) and buf2(j) will be used in exactly the same way as in LB.

An L_j/H_j pair, produced after the horizontal filtering of IPM(j), is either stored in a line-buffer

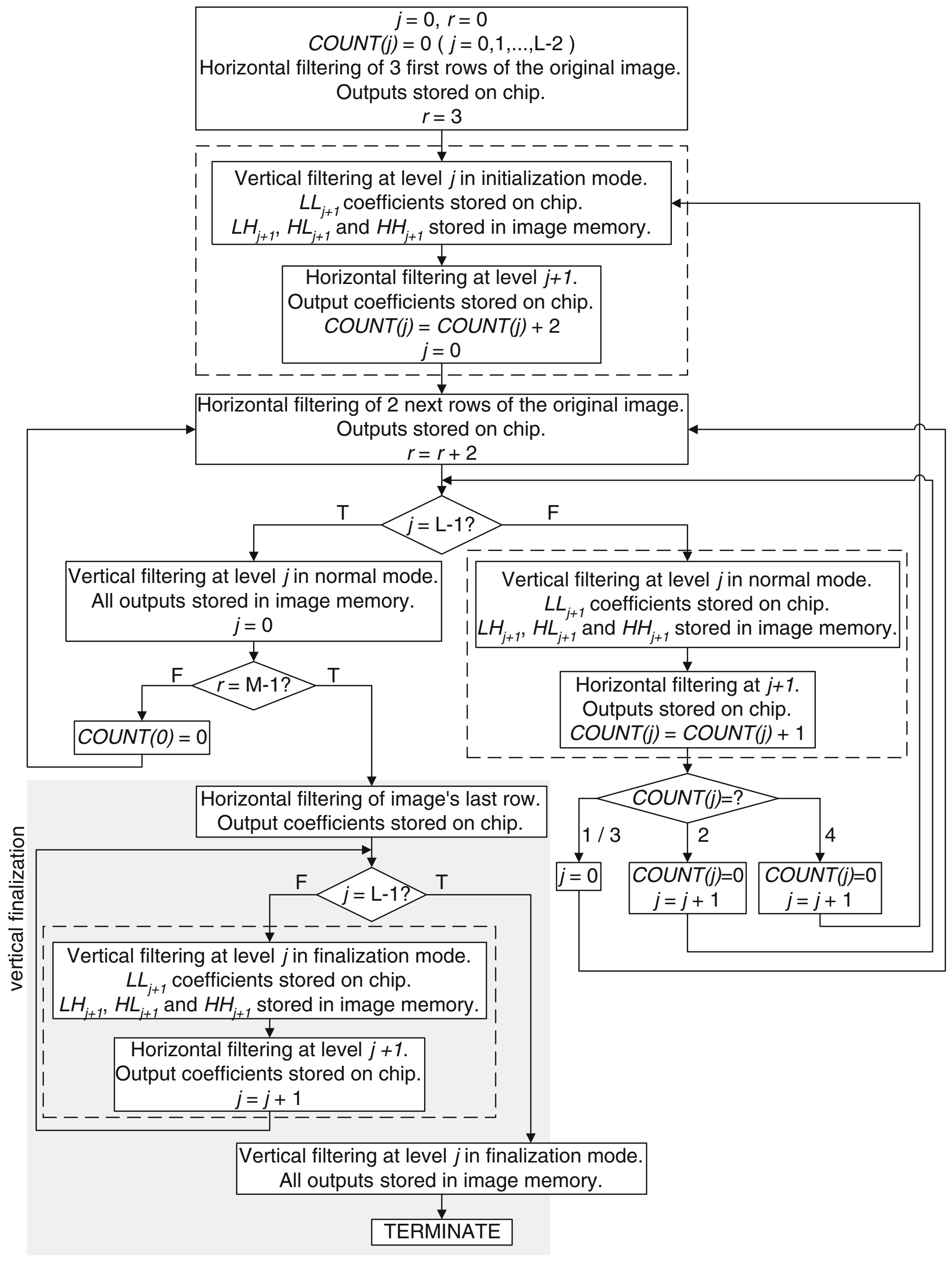
j = 0, r = 0
COUNT(j) = 0 (j = 0,1,...,L-2)
Horizontal filtering of 3 first rows of the original image. Outputs stored on chip. r = 3
Vertical filtering at level j in initialization mode. LLj+1 coefficients stored on chip. LHj+1, HLj+1 and HHj+1 stored in image memory.
Horizontal filtering at level j+1. Output coefficients stored on chip. COUNT(j) = COUNT(j) + 2 j = 0
Horizontal filtering of 2 next rows of the original image. Outputs stored on chip. r = r + 2
j = L-1?
T
F
Vertical filtering at level j in normal mode. All outputs stored in image memory. j = 0
r = M-1?
COUNT(0) = 0
Vertical filtering at level j in normal mode. LLj+1 coefficients stored on chip. LHj+1, HLj+1 and HHj+1 stored in image memory.
Horizontal filtering at j+1. Outputs stored on chip. COUNT(j) = COUNT(j) + 1
COUNT(j)=?
1 / 3
2
4
j = 0
COUNT(j)=0 j = j + 1
vertical finalization
Horizontal filtering of image's last row. Output coefficients stored on chip.
Vertical filtering at level j in finalization mode. LLj+1 coefficients stored on chip. LHj+1, HLj+1 and HHj+1 stored in image memory.
Horizontal filtering at level j +1. Output coefficients stored on chip. j = j + 1
Vertical filtering at level j in finalization mode. All outputs stored in image memory.
TERMINATE

Figure 13. Flowchart of the LB algorithm as implemented (r=current row, j=current level).

(for the odd rows of level j) or consumed at once (for the even rows of level j) to produce either an LL_{j+1}/LH_{j+1} or an HL_{j+1}/HH_{j+1} pair (Fig. 18). Hence, in the case of even rows, a vertical filtering action is undertaken after the generation of every L_j/H_j pair, and no supplementary line-buffer is needed. On the contrary, in Fig. 17, in the even rows of vertical filtering, a whole row of L_j and H_j coefficients had to be written in R(j), so that continuous horizontal filtering would be applied on it. Only after the completion of the horizontal filtering of the even row of level j, the vertical filtering would begin. As a result, the number of line buffers of BB is reduced by one, compared to that of LB.

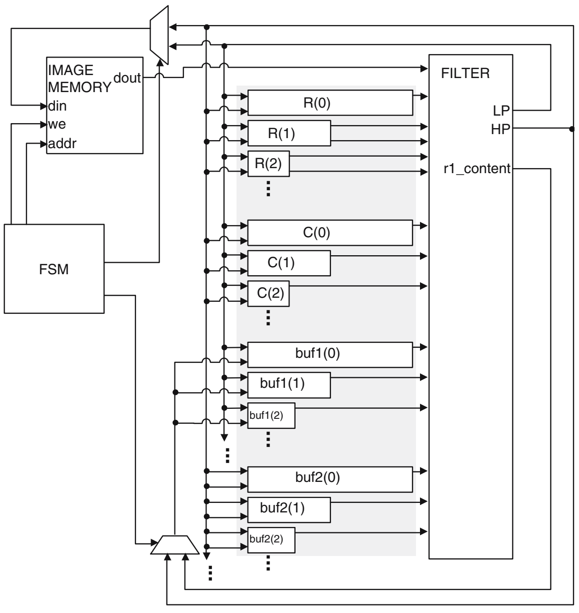

Figure 14. Block diagram of the LB architecture. The shaded area includes the on-chip buffers used in the architecture.

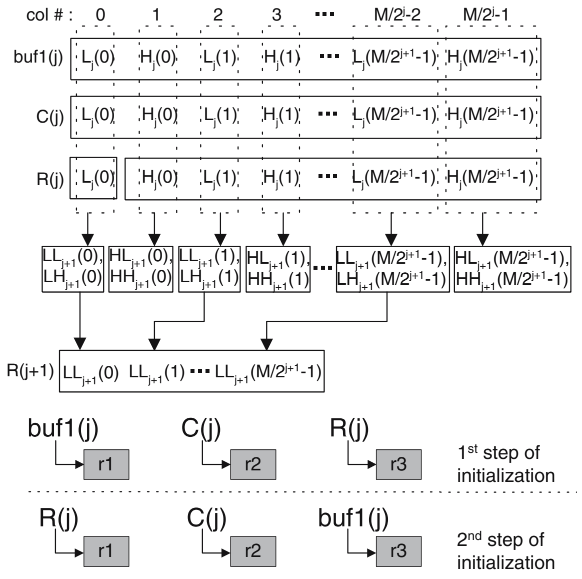
col # : 0 1 2 3 ... M/2^j-2 M/2^j-1
buf1(j)
C(j)
R(j)
L_j(0) H_j(0) L_j(1) H_j(1) ... L_j(M/2^{j+1}-1) H_j(M/2^{j+1}-1)
LL_{j+1}(0), LH_{j+1}(0)
HL_{j+1}(0), HH_{j+1}(0)
LL_{j+1}(1), LH_{j+1}(1)
HL_{j+1}(1), HH_{j+1}(1)
LL_{j+1}(M/2^{j+1}-1), LH_{j+1}(M/2^{j+1}-1)
HL_{j+1}(M/2^{j+1}-1), HH_{j+1}(M/2^{j+1}-1)
R(j+1) LL_{j+1}(0) LL_{j+1}(1) ... LL_{j+1}(M/2^{j+1}-1)
r1
r2
r3
1st step of initialization
2nd step of initialization

Figure 16. Vertical filtering in initialization mode in the LB architecture. The second step of initialization can be simplified avoiding the overwriting of r2 with the same value it already has, by forcing its *we* input to low. Also at the second step, r3 will be written with the previous value of r1.

The block diagram of the BB architecture is shown in Fig. 19. The on-chip memory needed for level j is illustrated in Fig. 20. The control logic, implemented with the FSM, is a lot more complicated in the case of BB, compared to that of LB. Mainly, it differs from the control logic presented in Fig. 13 in the following:

1. Successive columns of L_j and H_j coefficients are no longer filtered vertically in a successive manner. After a vertical filtering of an even column occurs, if the current level is not also the final one, LL_{j+1} is either written in bufH(j+1) or in IPM(j+1), depending on the current stage of horizontal filtering at level $j+1$. A single step of horizontal filtering at level $j+1$ may occur (depending on the horizontal filtering stage) interrupting the vertical filtering of level j.

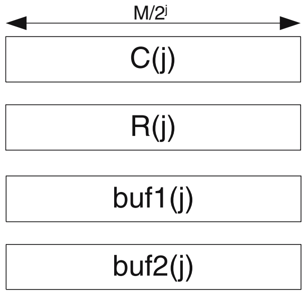

Figure 15. On-chip line buffers of level j, used in LB.

2. The horizontal filtering is no longer continuous. After a discrete step of horizontal filtering at level $j+1$, it should be decided if a vertical filtering at level $j+1$ can occur. If it does occur, a single step of horizontal filtering at level $j+2$ might follow, and so on. This domino effect in the worst case reaches the final level, imposing highly frequent interchanges between successive levels, that complicate the control logic.

As in LB, the filter-bank used in BB should make full use of the parallelism multi-port buffers offer. Thus, the version of Fig. 9 is used, so that the filter-bank operates in a single-input mode during the horizontal filtering at level 0 and functions in a multiple-input mode in any other case.

4. Results and Comparisons

The architectures were implemented in VHDL, synthesized with Synplify Pro 7.7, and placed and routed on Xilinx Virtex 4 XC4VLX15 FPGA, using Xilinx ISE v.8.1. In all of the implementations, the image memory shares the same clock with the rest of the system. This simplification renders our compari-

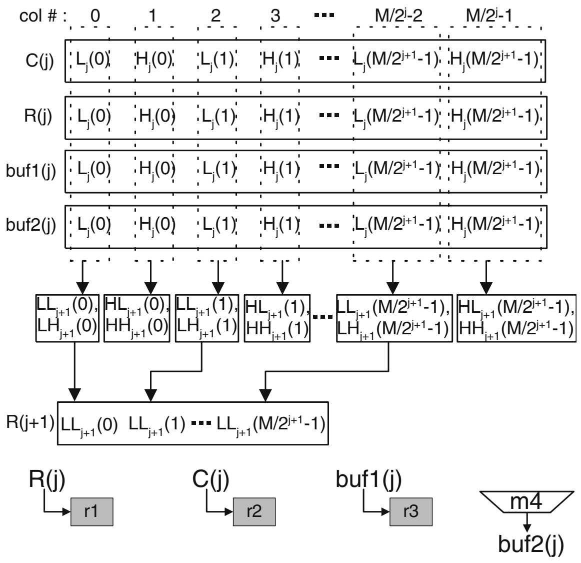

Figure 17. Vertical filtering in normal mode in the LB architecture. Three inputs are loaded in parallel into the FIFO, while buf2(j) passes through multiplexer m4 of the filter-bank.

son as generic as possible, since the image memory is sometimes on-chip (e.g. in large FPGA devices). Under this assumption, it should be noted that the computation

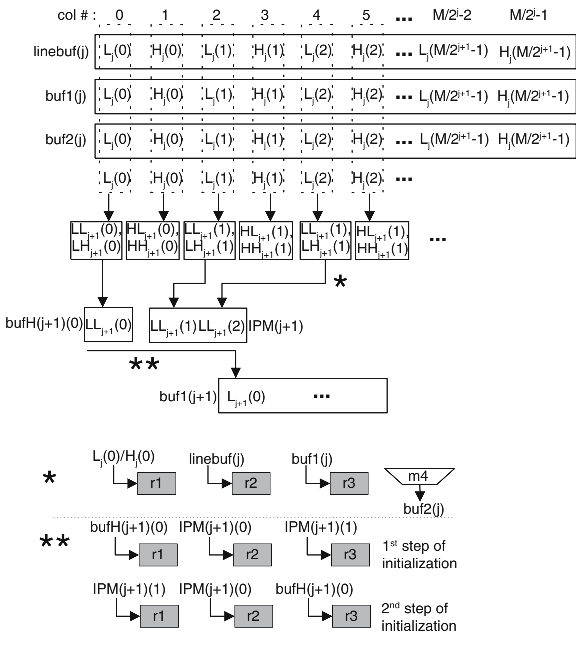

Figure 18. The normal-mode vertical filtering at level j, in BB, is followed by initialization-mode horizontal filtering at level $j+1$. Being at the beginning of the vertical initialization stage at level $j+1$, L_{j+1} is written in buf1(j+1).

schedules with larger traffic towards the image memory are favored, and no interface circuit is needed.

The current section presents a comparative analysis of the implementations, in terms of throughput, number of FPGA slices, memory requirements and energy consumption. These measurements are relative to the image size (M) and the number of decomposition levels (L).

4.1. Throughput

The RC, the LB and the BB operate on the XC4VLX15 device at 172.4, 113.6 and 117.6 MHz respectively. The three schemes have similar data paths: 174.6 MHz for the RC, 171.3 MHz for the LB and 170.9 MHz for the BB. However, the frequency varies because the critical path lies in the control path.

The LB obtains the highest throughput among the three schedules. This is due to the small number of clock cycles it requires to complete the 2D DWT (Table 1); it starts from 150,024 cycles (M=256, L=3), and reaches 2,374,740 cycles (M=1024, L=6). The throughput of LB reaches 757 frames/second (M=256, L=3) and drops to 47 frames/second (M=1024, L=6).

There is no great difference between the number of cycles needed for RC and BB. Specifically, RC needs 347,651–5,607,174 (Table 1), while BB requires 354,827–5,767,246 clock cycles (Table 1). However,

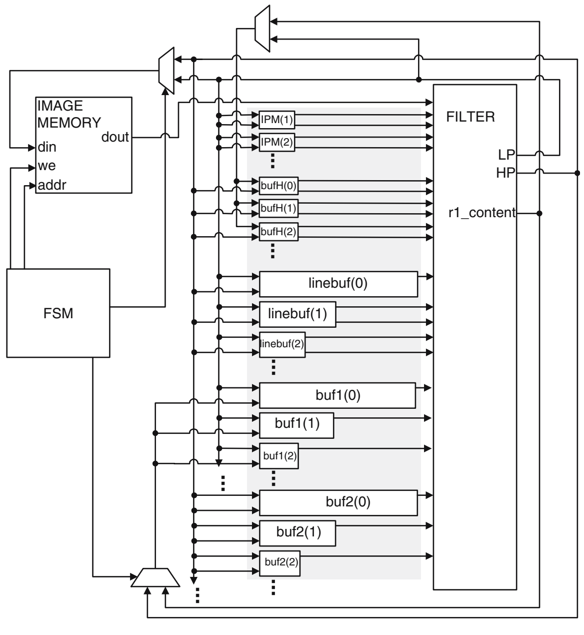

Figure. 19. Block diagram of the BB architecture. The shaded area includes the on-chip buffers used in the architecture.

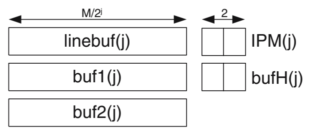

Figure 20. On-chip memory of level *j*, used in BB.

because the RC operates in higher frequency, its throughput is improved, resulting in the difference observed in Table 2. If the image memory operates in a smaller frequency than the rest of the system, BB will outperform RC.

The larger number of cycles in RC, compared to LB, is due to the fact that a single-port memory is used as the only source of inputs for RC. Thus, inputs enter the filter-bank in a serial manner and no parallelism is involved in the filtering operations. On the contrary, in LB and BB, two or three inputs might enter the filter-bank in parallel. Also, in RC the filter's output pair cannot be written in image memory in a single cycle; one of the outputs should be buffered to be written at the next cycle. The results that we got prove that using multi-port buffers, even with a single filter-bank, halves the number of cycles for the LB architecture. But this is not the case for the BB, even if multi-port buffers are also used to increase memory-access locality. This is due to the streaming nature of the operations taking part in LB, which is no longer the case for BB. Thanks to that, in LB at many points the next action is predetermined, for example the horizontal filtering is continuous and successive columns are successively filtered. As a result, many low level actions can occur in parallel, as it is pre-decided that they would not affect each other. Things are different for BB, since the control is deprived of such a streaming behavior and many options should be considered at a specific point, instead of following a predetermined route. As a result, in the case of BB, the number of cycles is increased, compared to LB.

Observing Table 2, one concludes that all three architectures can handle efficiently the image processing of still images, with minimal hardware cost—just one filter-bank with only one pipeline stage for the whole structure.

In order for a system to carry out video processing, the whole video-processing chain should operate at a speed of 30 frames/second. Therefore, a safety margin should be considered, to judge the video-processing capability of a system based on the results of Table 2. Observing Table 2, we can safely conclude that all three architectures can handle the video-processing of image sizes 256 and 512. The high throughput for such images would also allow an efficient operation of all three systems in a low-power mode. However, when it comes to image size 1,024, even though the throughput of the LB architecture still allows video-processing, this is no longer the case for the other two.

4.2. FPGA Slices

The FPGA slices used in RC are much fewer than in LB and BB (Table 3). This is due to the simplicity of the control associated with the RC algorithm. The number of slices for RC covers a range from 280 (M=256, L=3) up to 329 slices (M=1024, L=6). For LB and BB this range is 2,659–3,001 and 2,646–3,597 slices, respectively.

4.3. Memory Issues

The RC does not involve any on-chip buffers, contrary to LB and BB. Thus, in RC the image memory is the only source of inputs for the filter-bank. As a result, the number of image memory accesses is significantly larger in the RC case (Table 4). In the cases of LB and BB this number is the same, and does not vary when the number of levels varies, as it is, in both cases, equal to $2\times M^2$.

In LB and BB, the on-chip local memory of each level is accommodated in BRAMs (Fig. 21), gener-

Table 1. Number of clock cycles.

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	347,651	352,004	353,157	353,478	1,383,427	1,400,324	1,404,677	1,405,830	5,519,363	5,585,924	5,602,821	5,607,174
LB	150,024	150,988	151,280	151,380	592,904	596,364	597,328	597,620	2,357,256	2,370,316	2,373,776	2,374,740
BB	354,827	359,372	360,493	360,766	1,418,251	1,436,556	1,441,101	1,442,222	5,670,923	5,744,396	5,762,701	5,767,246

Table 2. Throughput (in frames/second).

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	495	489	488	487	124	123	122	122	31	30	30	30
LB	757	752	751	750	191	190	190	190	48	47	47	47
BB	331	327	326	326	82	81	81	81	20	20	20	20

ated by the Xilinx ISE Coregenerator, and registers. The bit-width used is 16 bits. The Virtex-4 BRAMs used are 18 K dual-port BRAMs.

To obtain maximum throughput, the buffers of the same type and of different levels can be grouped together in a single BRAM, if the space provided is enough. This way, during the vertical filtering of successive columns, R(j), C(j), buf1(j) and buf2(j) can feed the filter-bank simultaneously. At the same time, locations of buf1(j) and buf2(j), that have already been read, are overwritten with the new intermediate results, to be read at the next lifting step. Moreover, R(j) can be read while R(j+1) is written with the output of a previous column's vertical filtering. Therefore, four dual-port BRAMs should be used in the cases of image sizes 256 and 512. Four additional dual-port BRAMs will be needed for size 1,024, to accommodate the larger buffers of level 0. The number of BRAMs remains the same for 3, 4, 5 and 6 levels, to guarantee high throughput and enough space for the larger buffers of the lower levels.

In BB, buffers IPM and bufH of each level are implemented as two-word registers. Contrary to LB, the vertical filtering of successive columns is no longer successive. Thus, the constraints that guarantee maximum throughput are not so strict for BB. During vertical filtering, three filter inputs should be read simultaneously, thus, at least two dual-port BRAMs should be used. To provide enough memory space, 2, 3 and 6 BRAMs should be used for image sizes 256, 512 and 1,024, respectively. These choices, which also respect the minimum of two BRAMs, remain the same for 3, 4, 5 and 6 levels, since the larger buffers of the lower levels are the ones that determine the memory space needed.

5. Energy

In this section, we provide energy measurements for the implementations. First, we provide on-chip energy results, excluding the image memory. Afterwards, assuming an off-chip image memory and considering off-chip power estimates, we calculate the total energy required for the completion of the transform.

5.1. *On-Chip Energy*

To estimate the on-chip power consumption of each schedule, we have used the XPower tool offered by Xilinx ISE v.8.1. The estimated on-chip power dissipation of the RC is 214 mW, which does not notably alter as *M* or *L* vary. This is due to two factors: (a) the RC does not involve on-chip buffers, whose number would depend on the given parameters *M* and *L*, and (b) the variation in the number of slices for different pairs of these parameters is trivial (Table 3). This is no longer the case for the other two, where BRAMs are used, their number being directly dependent on *M* (Fig. 21), and where the number of slices varies to a larger extent, relative to the given parameters (Table 3). Thus, in the case of BB, the power consumption starts at 241 mW (*M*=256) and stretches to 268 mW (*M*=1024). The power consumption of LB is at relatively higher levels,

Table 3. Number of FPGA slices.

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	280	293	309	327	286	293	316	327	296	301	320	329
LB	2,659	2,792	2,805	2,881	2,851	2,860	2,901	2,967	2,962	2,965	2,970	3,001
BB	2,646	2,972	3,145	3,503	2,728	2,989	3,146	3,510	2,781	3,013	3,207	3,597

Table 4. Total number of accesses (contains both read and write accesses) of the image memory.

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	344,064	348,160	349,184	349,440	1,376,256	1,392,640	1,396,736	1,397,760	5,505,024	5,570,560	5,586,944	5,591,040
LB/BB	131,072	131,072	131,072	131,072	524,288	524,288	524,288	524,288	2,097,152	2,097,152	2,097,152	2,097,152

For LB and BB this number is the same, and does not vary as L varies, since it is, in both cases, equal to $2\times M^2$.

starting at 264 mW (M=256) and reaching 289 mW (M=1024). The smaller number of slices the LB uses, compared to BB, is overshadowed by the requirement for more BRAMs, and, therefore, the LB ends up with a higher power consumption.

The on-chip energy consumed for the execution of the transform, relative to M and L, is presented in Table 5. At a first glance, one notices the considerably larger amount of energy related to the BB, compared to the other two. This is due to the large number of cycles associated with the BB (Table 1). The number of cycles is once more the dominant factor in the energy calculations for the LB. That is, the significantly smaller number of cycles related to LB (Table 1) results in lower energy than RC, even though the RC is the less power-hungry of all.

5.2. *Total Energy*

We have presented power and energy results without considering the power associated with the image memory accesses. This power depends on whether the memory is on-chip or off-chip. Using an off-chip image memory is normally the only option, especially when working with large images or/and not very large FPGA devices. In that case, the power cost per memory access is significantly high, and depends on technology-related characteristics of the given memory.

We will consider a Synchronous DRAM as the off-chip image memory. Note here that the energy consumed at the off-chip interconnect is not included in our calculations of total energy. The energy of the off-chip interconnect would normally be proportional to the distance between the SDRAM and the FPGA device.

We have used the SDRAM system power calculator available from Micron Technologies [21], to calculate the off-chip power associated with a Micron 64 Mb x16 SDRAM. This tool bases the power calculation on a combination of SDRAM device specifications and usage conditions in the system environment. It, therefore, considers reliable estimates of the static and dynamic power, to calculate the total power dissipation of the memory.

The choice of SDRAM with respect to its clock frequency is based on the following two assumptions. Firstly, we have assumed that the image memory will operate at the same frequency as the rest of the system. As we have already mentioned, we have done this in order to make our comparison as generic as possible, since the image memory is sometimes on-chip. Secondly, we will operate the entire system at the frequency dictated by the on-chip design. Therefore, we can use an available SDRAM with clock frequency that is greater or equal to the frequency of the on-chip system.

As we have stated in Section 4.1, the RC, the LB and the BB have frequencies of 172.4, 113.6 and 117.6 MHz respectively. We have, therefore, considered SDRAM of clock frequency 183 MHz (speed grade=−55) for the case of RC, and 133 MHz (speed grade=−75) for LB and BB.

According to the estimates of the system power calculator, the off-chip power varies from 715.7 mW

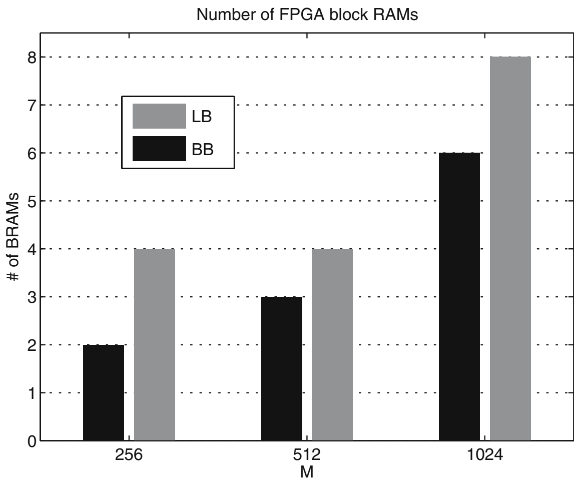

Figure 21. Number of BRAMs.

Table 5. On-chip energy consumption (in mJ).

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	0.432	0.437	0.438	0.439	1.717	1.738	1.743	1.745	6.850	6.933	6.954	6.959
LB	0.349	0.351	0.351	0.352	1.377	1.385	1.388	1.388	5.995	6.028	6.037	6.039
BB	0.727	0.736	0.738	0.739	3.002	3.040	3.050	3.052	12.918	13.09	13.127	13.138

to 720.3 mW for the case of RC, as the percentage of memory write/read cycles vary, for different values of *L* and *M*. Note that the variation is not as large, for different values of *M*, as one would expect observing Table 4. This is because, as *M* increases, the total number of cycles increases as well (Table 1), so the above percentages are maintained at similar levels. Due to the high cost per memory access that characterizes the off-chip case, the large number of accesses of the RC (Table 4), renders this schedule extremely demanding in terms of power.

Compared to the RC, the off-chip power of the LB and the BB is expected to be at lower levels, due to the lower number of memory accesses (Table 4). The off-chip power estimates vary between 549.7 mW and 559.7 mW for LB, and between 346 mW and 348.8 mW for BB. The lower off-chip power associated with the BB, compared to LB, is due to the higher percentage of memory write/read cycles over the total execution cycles. This is due to the fact that the number of clock cycles is much larger for BB (Table 1), while both the BB and the LB need M^2 read memory accesses and M^2 write memory accesses. Therefore the distribution of memory accesses is rather dense in LB, compared to BB. However, due to the significantly smaller number of clock cycles, required for its completion, the LB maintains the smallest energy when it comes to total consumption, as Table 6 shows.

The large number of memory accesses of the BB will keep the total energy consumption of BB at high levels, as it was the case with the on-chip energy. As a result, the total energy required for the BB to be executed is very close to that of the RC (Table 6), even though the power is significantly lower in BB.

Using an on-chip image memory would be in favor of the RC, since the power associated with every memory access will be much less. In this case, the BB will most certainly consume a lot more total energy than RC, as was the trend in Table 5. The LB will still maintain the lowest total energy.

5.3. Appropriate Optimizations

We have investigated the performance of the three schedules under the assumption of using a single-port image memory and a single filter-bank which is shared among all levels and stages of the transform. We will now briefly discuss which optimizations would be appropriate for each schedule to further improve its performance. Specifically, the objectives would be to increase (a) parallelism in data processing and (b) locality of data accesses.

In RC, as we have already stated, the lack of parallelism is due to the fact that the single-port image memory is the one and only source of the filter-bank's inputs. Therefore, using a multi-port image memory would multiply the performance by a factor equal to the number of memory ports. On the contrary, just increasing the number of filter-banks while sticking with a single-port memory would not

Table 6. Total energy consumption (in mJ).

M	256				512				1,024			
L	3	4	5	6	3	4	5	6	3	4	5	6
RC	1.876	1.899	1.904	1.906	7.486	7.576	7.599	7.604	29.908	30.266	30.356	30.379
LB	1.079	1.082	1.084	1.084	4.286	4.300	4.304	4.305	17.606	17.661	17.675	17.679
BB	1.778	1.795	1.799	1.800	7.206	7.274	7.291	7.295	29.733	30.018	30.089	30.106

An off-chip SDRAM is considered.

help, since there would not be enough on-chip information to process. As far as the locality of data accesses in RC is concerned, it could be improved by incorporating in the traditional bufferless architecture of Fig. 12 a single on-chip line-buffer. The inputs of the filter-bank would be read from this line-buffer in all levels. Therefore, its size should be equal to the largest dimension of the original image, to accommodate the large number of coefficients of the first level.

In the cases of LB and BB, the locality of data accesses is already satisfactory since image memory accesses are restricted to reading the original image and writing the final outputs. As far as the parallelism is concerned, since so much information exists in the on-chip buffers, the more filter-banks available to process it, the better. Thus, in this case increasing the number of filter-banks would vastly increase the parallelism in data processing, even if the image memory remains single-port.

6. Conclusion

The three major 5/3 lifting 2D DWT computation schedules have been compared in terms of throughput, area, memory and energy requirements on the Virtex-4 FPGA family. The conclusions of this work are the following.

The RC has by far the lowest hardware cost. Not only does it involve no on-chip buffering, but also the FPGA slices used are significantly fewer than in the cases of the other two. Due to its simplicity, it enjoys the highest frequency. However, the insufficient level of parallelism, associated with RC, increases the number of cycles required for the schedule to be completed. As a result, when it comes to throughput, it is outperformed by LB. Moreover, the reduced memory access locality, offered by RC, increases the number of memory accesses and the total energy to the highest level.

The LB requires the lowest number of cycles among the three schedules, thanks to, not only the parallelism achieved by using multi-port on-chip buffers, but also its streaming behavior. Due to the small number of cycles, this schedule also enjoys the highest throughput and the lowest on-chip and total energy consumption. Due to the use of on-chip buffers, the LB increases the memory-access locality, compared to RC, minimizing the number of image memory accesses.

The BB increases the memory-access locality, compared to RC, and achieves the same number of memory accesses as LB. To do this, the BB uses a smaller number of BRAMs than LB. However, it consumes more FPGA slices than LB for most sets of parameters. The BB is associated with the highest complexity, due to the lack of streaming behavior in the filtering operations. As a result, a significantly larger number of clock cycles is needed for the BB schedule to be completed. The large number of cycles results in high energy requirements: it is by far the most expensive when it comes to on-chip energy, and it is almost as expensive as RC when the total energy is considered, even though the memory accesses of BB are considerably fewer. The BB has the lowest throughput, due to control complexity, as well as to the frequency in which it operates, which is not as high as in the RC case.

Future work would involve extending the current comparative analysis by broadening the range of the comparison parameters. For instance, using dual-port image memory and more filters would be considered.

References

1. ISO/IEC FCD15444-1: 2000, “JPEG 2000 image coding system,” 2000.
2. ISO/IEC JTC1/SC29/WG11, FCD 14496-1, “Coding of moving pictures and audio,” 1998.
3. S. Mallat, “A theory for multiresolution signal decomposition: The wavelet representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, 1989, pp. 674–693.
4. I. Daubechies and W. Sweldens, “Factoring wavelet transforms into lifting steps,” *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, 1998, pp. 247–269.
5. K. Andra, C. Chakrabarti, and T. Acharya, “A VLSI architecture for lifting-based forward and inverse wavelet transform,” *IEEE Transactions on Signal Processing*, vol. 50, no. 4, 2002, pp. 966–977.
6. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, “Wavelet Transforms That Map Integers To Integers,” *Journal of Applied Computational Harmonics Analysis*, vol. 5, no. 3, 1998, pp. 332–369.
7. S. Dewitte and J. Cornelis, “Lossless Integer Wavelet Transform,” *IEEE Signal Processing Letters*, vol. 4, no. 6, 1997, pp. 158–160.
8. M. D. Adams and F. Kossentini, “Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis,” *IEEE Transanctions on Image Processing*, vol. 9, no. 6, 2000, pp. 1010-1024.
9. C. Chrysafis and A. Ortega, “Line-based, reduced memory, wavelet image compression,” *IEEE Transactions on Image Processing*, vol. 9, no. 3, 2000, pp. 378–389.
10. G. Lafruit, L. Nachtergaele, B. Vanhoof, and F. Catthoor, “The Local Wavelet Transform: a memory-efficient, high-speed architecture optimized to a Region-Oriented Zero-Tree coder,”

Integrated Computer-Aided Engineering, vol. 7, no. 2, 2000, pp. 89–103.
11. Cast, Inc., http://www.cast-inc.com.
12. Amphion Semiconductor, Ltd., http://www.amphion.com.
13. G. Dillen, B. Georis, J.-D. Legat, and O. Cantineau, "Combined Line-Based Architecture for the 5–3 and 9–7 Wavelet Transform of JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 9, 2003, pp. 944–950.
14. N. D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C. E. Goutis, "Evaluation of design alternatives for the 2-D-discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, 2001, pp. 1246–1262.
15. M. Weeks and M. Bayoumi, "Discrete Wavelet Transform: Architectures, Design and Performance Issues," *Journal of VLSI Signal Processing*, vol. 35, no. 2, 2003, pp. 155–178.
16. C. Chakrabarti, M. Vishwanath, and R. M. Owens, "Architectures for wavelet transforms: A survey," *Journal of VLSI Signal Processing*, vol. 14, no. 2, 1996, pp. 171–192.
17. Y. Andreopoulos, P. Schelkens, G. Lafruit, K. Masselos, and J. Cornelis, "High-level cache modeling for 2-D discrete wavelet transform implementations," *Journal of VLSI Signal Processing (special issue on Signal Processing Systems)*, vol. 34, no. 3, 2003, pp. 209–226.
18. K. Masselos, Y. Andreopoulos, and T. Stouraitis, "Performance comparison of two-dimensional discrete wavelet transform computation schedules on a VLIW digital signal processor," in *IEE Proceedings Vision, Image & Signal Processing*, vol. 153, no. 2, 2006, pp. 173–180.
19. C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform," *IEEE Transactions on Signal Processing*, vol. 53, no. 4, 2005, pp. 1575–1586.
20. M. Angelopoulou, K. Masselos, P. Cheung, and Y. Andreopoulos, "A Comparison of 2-D Discrete Wavelet Transform Computation Schedules on FPGAs," in *IEEE International Conference on Field Programmable Technology*, 2006, pp. 181-188.
21. Micron Technologies, "SDRAM System-Power Calculator," http://www.micron.com/support/designsupport/tools.

Maria E. Angelopoulou received the M.Eng. degree from the Department of Electrical and Computer Engineering, University of Patras, Greece in 2005. Since October 2005, she has been a PhD student in the Department of Electrical and Electronic Engineering, Imperial College London, UK. Her research interests include reconfigurable architectures, image and video processing. She received several awards from the Greek State Scholarships' Foundation and the Technical Chamber of Greece during her undergraduate studies. She is a member of the IEEE and of the Technical Chamber of Greece.

Konstantinos Masselos received a first degree in Electrical Engineering from University of Patras, Greece in 1994 and an MSc degree in VLSI Systems Engineering from University of Manchester Institute of Science and Technology, United Kingdom in 1996. In April 2000, he got a PhD degree in Electrical and Computer Engineering from University of Patras, Greece. His Ph.D research was related to high level low power design methodologies for multimedia applications realized on different architectural platforms. From 1997 until 1999 he was associated as a visiting researcher with the Inter-university Micro Electronics Centre (IMEC) in Leuven, Belgium, where he was involved in research related to the ACROPOLIS multimedia pre-compiler. Until 2004, he was with INTRACOM S.A, Greece where he was involved in the realization of wireless communication systems. In 2005, he joined as a lecturer the Department of Electrical Engineering and Electronics of Imperial College London. Since 2006, he is an Assistant Professor in the Department of Computer Science and Technology of University of Peloponnese, Greece and a visiting lecturer at Imperial College. His main interests include compiler optimizations, high level power optimization, FPGAS and reconfigurable hardware, and efficient implementations of DSP algorithms. He is a member of the IEEE.

Peter Y. K. Cheung (M'85SM'04) received the B.S. degree with first class honors from Imperial College of Science and Technology, University of London, London, U.K. in 1973. Since 1980, he has been with the Department of Electrical Electronic Engineering, Imperial College, where he is currently a Professor of digital systems and deputy head of the department. He runs an active research group in digital design, attracting support from many industrial partners. Before joining Imperial College he worked for Hewlett Packard. His research interests include VLSI architectures for signal processing, asynchronous systems, reconfigurable computing using FPGA, and architectural synthesis. He was elected as one of the first Imperial College Teaching Fellows in 1994 in recognition of his innovation in teaching.

Yiannis Andreopoulos (M '00) obtained the Electrical Engineering Diploma and the MSc in Signal Processing Systems from the University of Patras, Greece in 1999 and 2000, respectively. He obtained the PhD in Applied Sciences from the University of Brussels (Belgium) in May 2005, where he defended a thesis on scalable video coding and complexity modeling for multimedia systems. During his thesis work he participated and was supported by the EU IST-project MASCOT, a Future and Emerging Technologies (FET) project. During his post-doctoral work at the University of California Los Angeles (US) he performed research on cross-layer optimization of wireless media systems, video streaming, and theoretical aspects of rate-distortion-complexity modeling for multimedia systems. Since Oct. 2006, he is a Lecturer at the Queen Mary University of London (UK). During 2002-2003, he made several decisive contributions to the ISO/IEC JTC1/SC29/WG11 (Motion Picture Experts Group – MPEG) committee in the early exploration on scalable video coding, which has now moved into the standardization phase. In 2007, he won the “Most-Cited Paper” award from the Elsevier EURASIP Journal Signal Processing: Image Communication, based on the number of citations his 2004 article “In-band motion compensated temporal filtering” received within a three-year period.