

Evaluation of Design Alternatives for the 2-D-Discrete Wavelet Transform

Nikos D. Zervas, Giorgos P. Anagnostopoulos, Vassilis Spiliotopoulos, Yiannis Andreopoulos, and Costas E. Goutis

Abstract—In this paper, the three main hardware architectures for the 2-D discrete wavelet transform (2-D-DWT) are reviewed. Also, optimization techniques applicable to all three architectures are described. The main contribution of this work is the quantitative comparison among these design alternatives for the 2-D-DWT. The comparison is performed in terms of memory requirements, throughput, and energy dissipation, and is based on a theoretical analysis of the alternative architectures and schedules. Memory requirements, throughput, and energy are expressed by analytical equations with parameters from both the 2-D-DWT algorithm and the implementation platform. The parameterized equations enable the early but efficient exploration of the various tradeoffs related to the selection to the one or the other architecture.

Index Terms—Comparative study, 2-D-DWT, VLSI architectures.

I. INTRODUCTION

THE INHERENT time-scale locality characteristics of the discrete wavelet transforms (DWTs) has established them as powerful tools for numerous applications such as signal analysis, signal compression, and numerical analysis. This has lead numerous research groups to develop algorithms and hardware architectures to implement the DWT. In [1]–[4], VLSI architectures for the 1- and 2- DWT have been proposed. Additionally, comparisons among the architectures and scheduling algorithms for the DWT, regarding their efficiency when the DWT is mapped in custom VLSI architectures, has been performed in [5], [6].

Although the comparisons presented in [5] and [6] are enlightening, the related analysis is performed in an abstract level ignoring implementation platform parameters (e.g., memories' latency, number of ports, type of filters etc.) that can heavily affect the results of such a comparison. Additionally, no direct comparison in terms of energy efficiency has been attempted so far. Furthermore, the possible optimizations and their effect in the design parameters are not discussed by prior work. However, prior work has pointed that none of the alternative architectures has a clear lead in terms of either memory requirements or throughput or energy dissipation for all possible sets of parameters. Hence, the researcher or designer has not yet been provided with the an analytical comparison that will enable the

early and secure selection of one among the alternative architectures for the 2-D-DWT. In this paper, we attempt to fill this gap. Specifically, the main VLSI architectures for the 2-D-DWT are analytically described. Additionally throughput and memory minimization optimizations are presented and their effect is analyzed. The main contribution of this paper is the comparative study of the alternative architectures, which is based on the development of analytical equations for memory requirements, throughput, and energy. Analysis focuses on the forward 2-D-DWT. It is considered that comparison result are also valid for the inverse 2-D-DWT, since hardware architectures for the inverse 2-D-DWT use the same resources and a reversed control flow.

The rest of this paper is structured as follows. In Section II, the basic background needed to follow this paper is given. In Section III, the core structure of any architecture for the 2-D-DWT, namely the filters, are described and throughput optimizations are discussed. Section III also describes a memory minimization technique applicable in any architecture for the 2-D-DWT. In Sections IV–VI, the three architecture alternatives for the 2-D-DWT are analyzed. In Section VII, we compare the alternative architectures, while in Section VIII, some conclusions are drawn.

II. BASIC BACKGROUND

In this section, the necessary background to follow this paper is reviewed. Specifically, Section II-A briefly describes the 1- and 2-D DWT decomposition, while Section II-B presents the energy model used for the characterization of the alternative architectures.

A. The DWT

The 1-D-DWT can be viewed as the multiresolution decomposition of a sequence [7]. It takes a length N sequence $IN[n]$, and generates an output sequence of length N . The output is a multiresolution representation of $IN[n]$. The highest resolution level is of length $N/2$, the next resolution level is of length $N/4$, and so on. We denote the number of frequencies or resolutions levels or levels with the symbol L . The 1-D-DWT filter bank structure, realizing the 1-D-DWT dyadic decomposition, is illustrated in Fig. 1. Typically, $h[n]$ and $w[n]$ of Fig. 1 are convolutional Quadrature Mirror Filters (QMF). We denote N_H and N_L the number of taps of the high-pass ($h[n]$) and low-pass ($w[n]$) filters respectively, and define $N_W = \max\{N_H, N_L\}$. For the sequence of low- and high-frequency coefficients of decomposition layer j we use the symbols $L_j[n]$ and $H_j[n]$, respectively. Hence, using mathematical notations, the low- and

Manuscript received November 17, 2000; revised October 7, 2001. This paper was recommended by Associate Editor R. Chandramouli.

N. D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, and C. E. Goutis are with the VLSI Design Laboratory, Department of Electrical and Computer Engineering, University of Patras, Patras GR-26500, Greece (e-mail: zervas@ee.upatras.gr; anagnost@ee.upatras.gr; bspili@ee.upatras.gr; goutis@ee.upatras.gr).

Y. Andreopoulos is with Pleinlaan2, Department ETRO, Vrije Universiteit Brussel, Brussels, Belgium (e-mail: vandreop@etro.vub.ac.be).

Publisher Item Identifier S 1051-8215(01)11036-0.

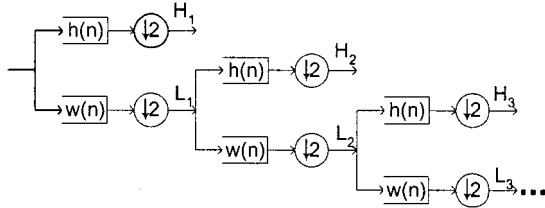


Fig. 1. 1-D-DWT decomposition.

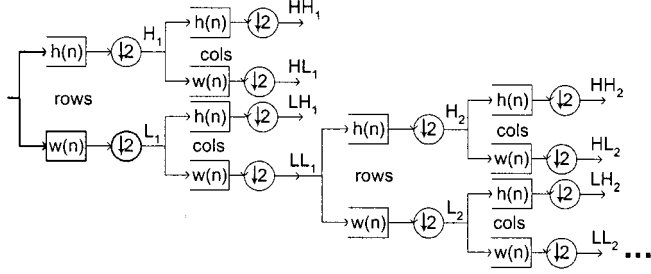


Fig. 2. 2-D-DWT decomposition.

high-frequency coefficients of decomposition level j are computed as follows:

$$L_{j+1}[n] = \sum_{i=0}^{N_L-1} w[i] \times L_j[2n - i] \quad (1)$$

$$H_{j+1}[n] = \sum_{i=0}^{N_H-1} h[i] \times L_j[2n - 1 - i] \quad (2)$$

where $j = \{0, 1, \dots, L - 1\}$, $n = \{0, 1, \dots, N/2^{j+1} - 1\}$, and $L_0[n] = IN[n]$.

The 2-D-DWT binary-tree decomposition is illustrated in Fig. 2. For each level, the input signal is filtered along rows and the resulted signal is filtered along columns. In this way, the 2-D decomposition of an input signal $IN[M][N]$, with M columns and N rows, is described by the following equations:

$$L_{j+1}[\text{row}][m] = \sum_{i=0}^{N_L-1} w[i] \times LL_j[\text{row}][2m - i] \quad (3)$$

$$H_{j+1}[\text{row}][m] = \sum_{i=0}^{N_H-1} h[i] \times LL_j[\text{row}][2m - 1 - i] \quad (4)$$

$$LL_{j+1}[n][\text{col}] = \sum_{i=0}^{N_L-1} w[i] \times L_j[2n - i][\text{col}] \quad (5)$$

$$LH_{j+1}[n][\text{col}] = \sum_{i=0}^{N_H-1} h[i] \times L_j[2n - 1 - i][\text{col}] \quad (6)$$

$$HL_{j+1}[n][\text{col}] = \sum_{i=0}^{N_L-1} w[i] \times H_j[2n - i][\text{col}] \quad (7)$$

$$HH_{j+1}[n][\text{col}] = \sum_{i=0}^{N_H-1} h[i] \times H_j[2n - 1 - i][\text{col}] \quad (8)$$

where $j = \{0, 1, \dots, L - 1\}$, $\text{row} = \{0, 1, \dots, N/2^j - 1\}$, $m = \{0, 1, \dots, M/2^{j+1} - 1\}$, $\text{col} = \{0, 1, \dots, M/2^{j+1} - 1\}$, $n = \{0, 1, \dots, N/2^{j+1} - 1\}$, and $LL_0[n][m] = IN[n][m]$.

In the rest of this document, we use the term *layer* to indicate both intermediate and output signals, i.e., L_j , H_j , LL_j , LH_j , HH_j , and HL_j , while *level* is used for each decomposition stage.

B. Energy Model

For the energy characterization of the alternative hardware architectures for the 2-D-DWT, only energy consumed due to data storage and transfers is taken into account. This suffices for the purposes of this paper for two reasons.

- 1) In hardware implementation of data-intensive algorithms, such as the 2-D-DWT, the energy dissipation due to data storage and transfers forms the dominant component (up to 80%) of the total power budget [8]. It is indicative that a transfer to/from an on-chip memory consumes 4–10 times more power than one addition, while an off-chip accesses requires 10–100 times more power than an on-chip access [8].
- 2) The different hardware architectures perform exactly the same number of filtering operations. Thus, it can be said that energy consumed to arithmetic operations is a common cost for all architectures.

The energy dissipated on the memory hierarchy is approximated by the energy dissipation due to on-chip memory accesses plus the energy dissipation due to off-chip memory accesses

$$E_{\text{MEM}} = \sum_i E_{\text{On_Chip_Mem}_i} + \sum_i E_{\text{Off_Chip_Mem}_i} \quad (9)$$

The energy consumed upon on-chip interconnect (buses) is much smaller than the internal power consumption of on-chip memories; thus, the energy cost of an on-chip memory transfer is approximated by the energy cost of the memory access itself. The energy consumed on accesses to the on-chip memories is estimated using the model presented by Landman in [9] and [10]. According to this model, the energy dissipated on memory accesses is a function of the memory size in terms of stored words, the number of bits per stored word, the number of accesses, the technology, and the number and the type (R or R/W) of ports. It is assumed that the energy is linearly proportional to the number of accesses, and sub-linear to memory size. We also assume supply voltage for all architectures. Thus, the energy dissipation due to on-chip memory accesses is given as

$$E_{\text{On_Chip}} = N_{\text{AccessesOn_Chip}} \cdot f(N_{\text{words}}, N_{\text{bits}}, N_{\text{ports}}) \quad (10)$$

For a given supply voltage, the function f of (10) determines the relation between the memory energy consumption and the memory size and depends only on technology. Such a function is described in [8]–[10] and is used for the estimation of memory access energy cost in this paper.

During an off-chip memory access, power is consumed by the bus driver, the memory, the processing element(s), chip I/O pins (bonding wires and pads), the bus wires, and the memory banks. High-level accurate estimation of the effective capacitance corresponding to each one of the above sources of power consumption is very difficult to be made. However, a rough but still useful

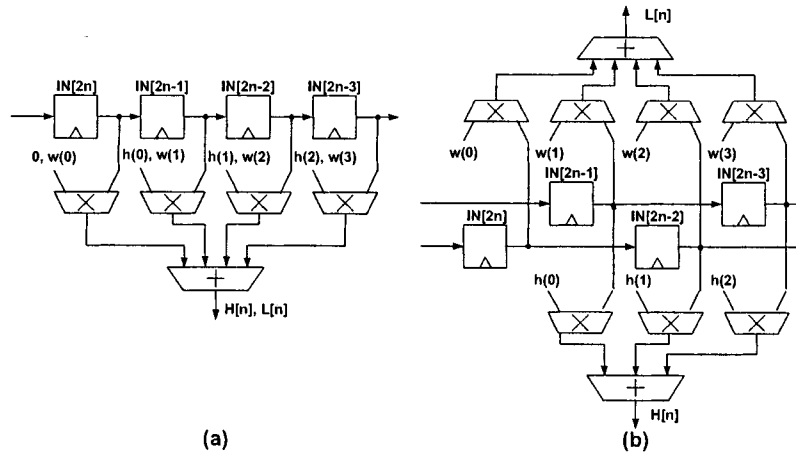


Fig. 3. Parallel filters. (a) Conventional. (b) Throughput optimized.

(for the aim of comparison) estimate can be acquired by considering typical values for the effective capacitance corresponding to each one of the above factors [8]. More precisely, the off-chip memories are assumed to be the most power conscious ones, i.e., low-power SRAM [11]. The internal power consumption of the off-chip memories is also modeled by (10). For the I/O pins, the bus driver, and the bus wires, a capacitance of 40 pF per bus line is assumed. It is also assumed that in average half of the off-chip bus (bit) lines make a transition per off-chip memory access. Thus the effective capacitance can be derived by multiplying the number of bus-lines (word-length) with the half of the 40 pF. This results in the following formula:

$$E_{\text{Off_Chip}} = [N_{\text{bits}} \cdot N_{\text{ports}} \cdot 10^{-11} \cdot V_{DD}^2 + f(N_{\text{words}}, N_{\text{bits}}, N_{\text{ports}})] \cdot N_{\text{AccessesOff_Chip}} \quad (11)$$

III. COMMON ISSUES

This paper does not cover all architectures proposed in the past, but focuses on these that are likely to be implemented in real-life designs. So, to be realistic, we consider RAM-based architectures that use parallel filters. We choose these architectures since they offer the highest regularity/density of storage and scale more easily, compared to architectures based on systolic or semi-systolic routing [3], [5]. Finally, we choose parallel filters because: 1) they offer a throughput equal to one output per cycle and 2) they can be pipelined at any level, unlike serial filters.

High throughput is imposed by the application domain of the 2-D-DWT, namely image/video compression, in which real-time operation is typically required. We reiterate here that the 2-D-DWT is a computational intensive algorithm, which has a complexity in terms of filtering operations in the order of $O(kN \times M)$, where N , M are the input's dimensions and k a constant. High throughput is also significant for low-power application, where it can be traded for reduced power supply [12]. Of course, the high throughput of parallel filters comes at the expense of a greater number multipliers. Although this is true, recall that unlike the 1-D case, in architectures for the 2-D-DWT, it is the storage that dominates on design's size and complexity, not the number of multipliers [5].

TABLE I
COMPARISON OF PARALLEL FILTER ARCHITECTURES FOR THE DWT

Filter Arch.	FPGA (xcv300-5, 0.22 μ m)		ASIC (AMS 0.6 μ m)	
	Delay (ns)	Area (CLBs)	Delay (ns)	Area (mil ²)
Conventional	28	1117	63.17	7203
Conv. + OSR	26	735	62.62	5474
Throughput Opt.	24	1856	61.98	12638
Thr. Opt. + OSR	21	704	47.84	6262

(OSR: operation strength reduction)

A. Parallel Filter Architectures for the DWT

As far as parallel filters for the DWT are concerned, the conventional and a throughput-optimized hardware architecture are studied. Conventional architecture consists of an input FIFO with width equal to N_W . Additionally, the same N_W multipliers are used for the computation of the low- and high-frequency outputs. The conventional architecture implements (1) and (2) in an interleaved manner. Specifically, for the even clock cycles, the multipliers are fed with the constant coefficients of the low-pass filter, while for the odd cycles, the same multipliers are fed with the constant coefficients of the low-pass filter.¹ In this way, a pair of high- and low-frequency coefficients is produced each two clock cycles. Throughput-optimized architecture consists of a modified FIFO that receives two input pairs per clock cycle and a separate data-path for the low-pass and high-pass filtering. Thus, throughput-optimized architecture produces a pair of high and low coefficients each clock cycle. Fig. 3 illustrates the conventional and the throughput optimized parallel filter for a 4/3 DWT.

Although it is expected that the conventional architecture occupies less area than the throughput-optimized, this is not always the case. This is because with the throughput-optimized architecture, the efficient application of an additional optimization is enabled. Specifically, multiplication among a variable (input sample) and a constant can be easily reduced to a number of shift and add operations, resulting this way in a much smaller implementation [13]. For example, a multiplication times 3 is reduced to a left shift by one and an increment by one. In Table I, the area

¹Obviously, the same holds for either (3) and (4), (5) and (6), or (7) and (8).

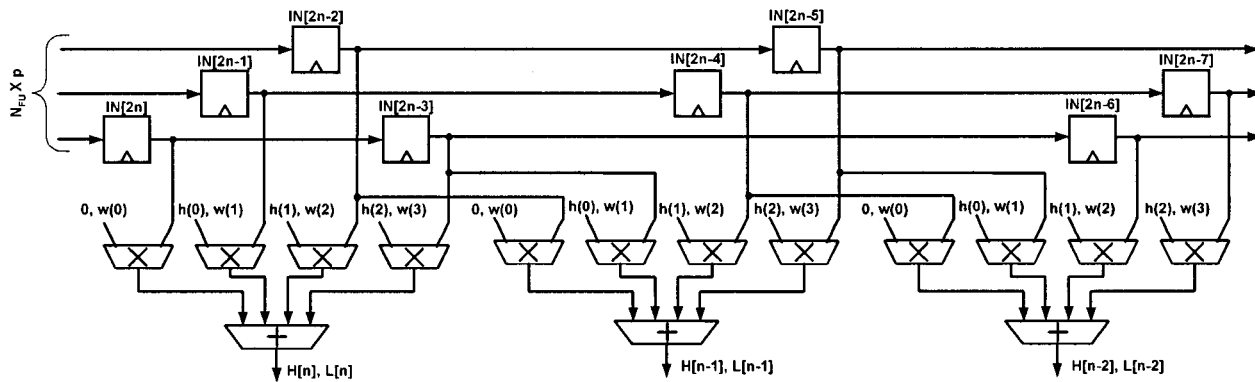


Fig. 4. Parallelization of filtering operations performed along a 1-D input sequence.

occupied by the nonpipelined conventional and throughput-optimized architectures, with and without operation reduction, for the widely used 9/7 filter is given. It can be observed that the throughput-optimized architecture with operation strength reduction is in all cases faster and, in most cases, smaller than its nonoptimized counterparts.

Another difference among the two alternative parallel filter architectures for the DWT is that the throughput-optimized architecture receives two inputs in parallel. For DWT RAM-based architectures, this imposes to use dual-port memories and wider input-data buses. This is expected to slightly increase the energy budget for the computation of the DWT, since multiport memories consume more energy per access than single-ported ones.

The following analysis of different 2-D-DWT hardware architectures uses the parameter p to model the selection among conventional and throughput-optimized parallel filter. Parameter p equals the number of input per cycle fed to the parallel filter. Thus, $p = 1$ indicates the usage conventional, while $p = 2$ indicates the usage throughput-optimized parallel filter.

B. Parallelization of Filtering Operation

According to the decomposition of Fig. 2, the 2-D-DWT is computed by applying the high- and low-pass filters along row and columns of a layer. To speed up the process of filtering along a 1-D input sequence, a linear array of filters can be used. Due to down-sampling by two, two new input coefficients are required to produce the next low- and high-frequency coefficient. Thus, to perform N_{FU} successive filtering operations along a 1-D input sequence, $N_W + 2 \times (N_{FU} - 1)$ input coefficients are needed. Hence, such a linear array of N_{FU} parallel filters requires an input FIFO of width $N_W + 2 \times (N_{FU} - 1)$. The first filter in the array receives input from position 0 up to $N_{FU} - 1$ of the FIFO, the next filter receives input from position 2 up to $N_{FU} + 1$, and so on. In Fig. 4, a linear array of three conventional parallel filters implementing a 4/3 DWT is illustrated. Obviously, the usage of such a filter array is rational only if $N_{FU} \times p$ input coefficients can be fetched/stored in one cycle. In other words, RAMs feeding input coefficients to and storing output coefficients from such an array should have $N_{FU} \times p$ read and write ports. Under this constraint, a utilization factor very close to 100% is realized when such an array is used to filter a 1-D input sequence. We note that realistic values of the product $N_{FU} \times p$ are in the range of 1–4.

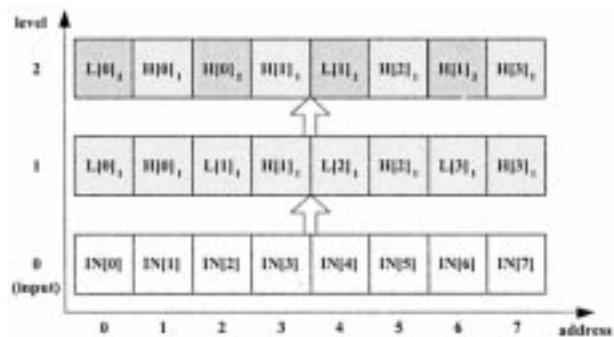


Fig. 5. In-place mapping for the 1-D-DWT.

The usage of a linear array of filters is not the only way to succeed parallelism for the computation of the 2-D-DWT. Another option is to employ parallelism among filtering operations of different layers of the 2-D-DWT. In this paper, this form of filtering operations parallelism is not studied, since this approach requires a more complex control and does not allow for 100% utilization factor in the general case.

The following analysis of different 2-D-DWT hardware architectures uses the parameter N_{FU} to model the number of parallel filters in the linear array.

C. In-Place Mapping for the 1- and 2-D DWT

Typically in any architecture for the 1-D or the 2-D DWT, two different memory blocks are allocated to store the input and the output of the transform. We remind that the input and the output of the transform, and thus also the corresponding memory blocks, are of equal size. In this subsection, we describe a in-place mapping scheme that allows to perform the 1-D or 2-D DWT using only one of these memory blocks. The key concept is simple: *Store filtering outputs in-place of no-longer needed filtering inputs.*

For example, consider the 1-D-sequence of Fig. 5 and assume that input coefficients are fed from the input memory to the filter from a FIFO, which we name filtering FIFO (FF). Additionally assume that input memory stores coefficient $IN(0)$ at address 0, coefficient $IN(1)$ at address 1, and so on. The pair of coefficients $L_1[0]$, $H_1[0]$ is produced by filtering the three first input coefficients after performing a symmetrical mirroring. Since input coefficients $IN[0]$ and $IN[1]$ are currently

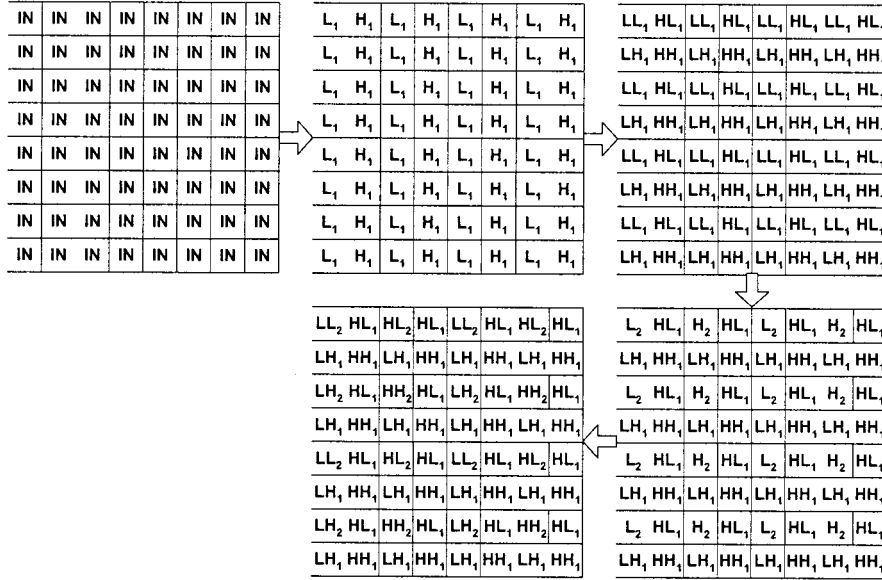


Fig. 6. In-place mapping for the 2-D-DWT.

in the FF and will not be fetched again from the IN_Mem , we can store $L_1[0]$, $H_1[0]$ in their place (addresses 0 and 1). In the same way, coefficients $L_1[i]$ and $H_1[i]$, can be stored at addresses $2 \cdot i$ and $2 \cdot i + 1$ respectively. A direct effect of this in-place mapping is that after the completion along the input, the low-frequency coefficients of level 1 that will be consumed to produce the coefficients of level 2 are not stored in consecutive addresses in the memory. Specifically, the low-frequency coefficients of level j ($L_j[i]$) are stored in the addresses $L_j[i] \sim IN[2^j \cdot i]$, while the high-frequency coefficients of level j ($H_j[i]$) are stored in the addresses $H_j[i] \sim IN[2^j \cdot i + 2^{j-1}]$ ($i = 0, 1, \dots, N/2^j - 1$ and $j = 1, 2, \dots, L$).

For the 2-D-DWT, consider now the input and decomposition layers as 2-D arrays of coefficients. In an analogous way, the coefficients of *row*, *col* of decomposition layers L_j and H_j are stored in the addresses

$$\begin{aligned} L_j[\text{row}][\text{col}] &\sim IN[2^{j-1} \cdot \text{row}][2^j \cdot \text{col}] \\ H_j[\text{row}][\text{col}] &\sim IN[2^{j-1} \cdot \text{row}][2^j \cdot \text{col} + 2^{j-1}] \end{aligned}$$

where $j = 1, 2, \dots, L$, $\text{row} = 0, 1, 2, \dots, N/2^{j-1} - 1$, and $\text{col} = 0, 1, 2, \dots, M/2^j - 1$.

The coefficients of *row*, *col* of decomposition layers HL_j , LH_j , LL_j , and HH_j are stored in the addresses

$$\begin{aligned} LL_j[\text{row}][\text{col}] &\sim IN[2^j \cdot \text{row}][2^j \cdot \text{col}] \\ LH_j[\text{row}][\text{col}] &\sim IN[2^j \cdot \text{row} + 2^{j-1}][2^j \cdot \text{col}] \\ HL_j[\text{row}][\text{col}] &\sim IN[2^j \cdot \text{row}][2^j \cdot \text{col} + 2^{j-1}] \\ HH_j[\text{row}][\text{col}] &\sim IN[2^j \cdot \text{row} + 2^{j-1}][2^j \cdot \text{col} + 2^{j-1}] \end{aligned}$$

where $j = 1, 2, \dots, L$, $\text{row} = 0, 1, 2, \dots, N/2^j - 1$, and $\text{col} = 0, 1, 2, \dots, M/2^j - 1$. An example of 2-D-DWT in-place mapping for input with size 8×8 is illustrated in Fig. 6.

Note that the addressing expressions required for the in-place mapping can be implemented in hardware using very simple

structures, since they consist of multiplications among indexes and a power of two (easily implemented by shifting operations) and increments by one. Although this is true, this additional hardware consumes additional energy. However, this energy penalty is insignificant since addressing is responsible for only a very small fraction of the total energy budget of architectures for the 2-D-DWT, which is dominated by the energy dissipation due data storage and transfers.

IV. ARCHITECTURE I: LEVEL-BY-LEVEL

The level-by-level architecture is the straightforward implementation of the 2-D decomposition of Fig. 2. Specifically, an input image is scanned in a row-by-row manner and filtering along layers is not interleaved. This means that, for each level, the filtering along columns is performed after the completion of the filtering along rows. Furthermore, the filtering of level j is initiated after the completion of filtering at level $j - 1$. The filtering operation schedule is described by the pseudo-code of Fig. 7², while the hardware architecture is illustrated in Fig. 8. It is noted that initialization and finalization process (needed at the image limits) is ignored in Fig. 7, and throughout this paper, for the sake of simplicity.

The memory Img_M (Fig. 8) initially stores the input image. Thus, the size of Img_M in terms of coefficients is

$$Img_M_Size = N \cdot M. \quad (12)$$

A linear array of N_{FU} parallel filters is used to perform the necessary filtering operations. The usage of such an array is sensible only if Img_M has $N_{FU} \times p$ read and write ports. Each layer's coefficients after their production are written back to Img_M according to the in-place mapping scheme described in Section III-C. It must be stressed here that with this architecture, it is meaningless to introduce local memories to store intermediate results (i.e., coefficients of layers H_j , L_j , and LL_j). This

²Routines `filter_row`, and `filter_col` are valid for $N_{FU} \cdot p = 2$.

```

begin{2D Level-by-Level Algorithm}
  for(j=1 to LEVELS)
    for(row=0 to N/2j-1)
      filter_row(row, j);
    for(cols=0 to M/2j-1)
      filter_col(col, j);
    end{2D Level-by-Level Algorithm}

filter_row(row, j)
begin{filter_row}
  for(col=0 to M/2j-1)
    /* Lj+1, Hj+1 production*/
    Img_M[2j-1 *row][2j *col], Img_M[2j-1 *row][2j *(col+1)] -> FF;
    filter(FF) -> Img_M[2j-1 *row][2j *col], Img_M[2j-1 *row][2j *(col+1)];
  end{filter_row}

filter_col(col, j)
begin{filter_col}
  for(row=0 to N/2j-1)
    /* LLj+1, LHj+1 production*/
    Img_M[2j-1 *row][2j *col], Img_M[2j-1 *(row+1)][2j *col] -> FF;
    filter(FF) -> Img_M[2j-1 *row][2j *col], Img_M[2j-1 *(row+1)][2j *col];
    /* HHj+1, HLj+1 production*/
    Img_M[2j-1 *row][2j *col + 2j-1], Img_M[2j-1 *(row+1)][2j *col + 2j-1] -> FF;
    filter(FF) -> Img_M[2j-1 *row][2j *col + 2j-1], Img_M[2j-1 *(row+1)][2j *col + 2j-1];
  end{filter_col}
end{filter_row}

```

Fig. 7. Level-by-level algorithm.

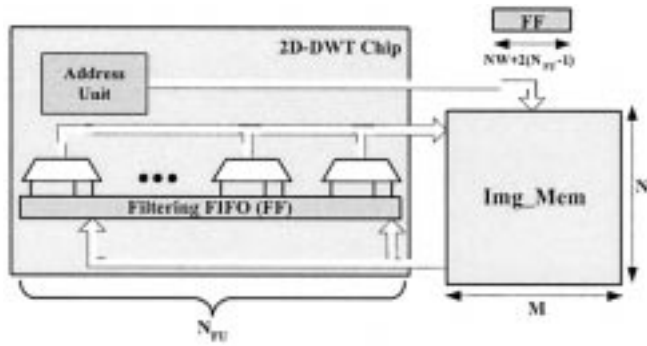


Fig. 8. Level-by-level architecture.

is due to the large size that such memories would have. For example, to store the coefficients of layers H_1 and L_1 requires a storage mean equal to the initial image size.

A. Number of Memory Accesses

Under the constraint related to the number of ports of Img_M , the number of read (write) accesses to Img_M required to perform the 2-D-DWT with the level-by-level approach is found as follows. To produce the H_j, L_j layers, we need to read all the coefficients of layer LL_{j-1} . Layer LL_{j-1} has a size of $(N/2^{j-1}) \cdot (M/2^{j-1})$. Since on each access $N_{FU} \cdot p$ coefficients are read (written) in parallel, the production of H_j, L_j layers requires $N \cdot M / (2^{2(j-1)} N_{FU} \cdot p)$ read (write) accesses. The size of layer L_j is $(N/2^{j-1}) \cdot (M/2^j)$, hence to produce the layers LL_j and LH_j , $N \cdot M / (2 \cdot 2^{2(j-1)} N_{FU} \cdot p)$ read (write) accesses are needed. In the same way, to produce HL_j and HH_j layers, another $N \cdot M / (2 \cdot 2^{2(j-1)} N_{FU} \cdot p)$ read (write) accesses are needed. Thus, for L levels of decomposition, the number of read (write) accesses is

$$\begin{aligned}
 N_Read_{Img_M} &= N_Writes_{Img_M} \\
 &= \frac{N \cdot M}{N_{FU} \cdot p} \sum_{j=1}^L \frac{2}{2^{2(j-1)}} \\
 &= \frac{8}{3} \left(1 - \frac{1}{4^L}\right) \frac{N \cdot M}{N_{FU} \cdot p}. \quad (13)
 \end{aligned}$$

B. Throughput

To come up with a formula for the throughput of the level-by-level approach, we need to define an extra parameter, namely the latency of Img_M . In this way, we name t_{Img_M} the number of clock cycles (latency) per Img_M access (read or write). Now, if we assume 100% utilization of all filters in the linear array, then the number of clock cycles (throughput) needed to perform the 2-D-DWT is

$$\begin{aligned}
 N_Cycles &= N_Reads_{Img_M} \cdot t_{Img_M} \\
 &= \frac{8}{3} \left(1 - \frac{1}{4^L}\right) \frac{N \cdot M \cdot t_{Img_M}}{N_{FU} \cdot p}. \quad (14)
 \end{aligned}$$

Finally, throughput in terms of input coefficients processed per second is

$$\begin{aligned}
 \text{Throughput} &= \frac{f_{clk}}{N_Cycles} \cdot N \cdot M \\
 &= f_{clk} \cdot \left[\frac{8}{3} \left(1 - \frac{1}{4^L}\right) \frac{t_{Img_M}}{N_{FU} \cdot p} \right]^{-1} \quad (15)
 \end{aligned}$$

where f_{clk} is the clock frequency.

C. Energy

Img_M can be stored either off-chip or on-chip, with respect to integration technology capabilities and image dimensions. For the case that Img_M is stored on-chip, the energy consumption of the level-by-level architecture is estimated using (11). Replacing in (11) the number of words with $N \cdot M$, the number of ports with $N_{FU} \cdot p$, the number of bits per word with d , and the number of accesses with $2 \times (13)$ results in the formula

$$\begin{aligned}
 E &= [d \cdot N_{FU} \cdot p \cdot 10^{-11} \cdot V_{DD}^2 \\
 &\quad + f(N \cdot M, d, N_{FU} \cdot p)] \cdot \frac{16}{3} \left(1 - \frac{1}{4^L}\right) \frac{N \cdot M}{N_{FU} \cdot p}. \quad (16)
 \end{aligned}$$

If Img_M is integrated on-chip, then the energy consumption of the level-by-level architecture is estimated using (10), and thus, the above equation is reduced to

$$E = f(N \cdot M, d, N_{FU} \cdot p) \cdot \frac{16}{3} \left(1 - \frac{1}{4^L}\right) \frac{N \cdot M}{N_{FU} \cdot p}. \quad (17)$$

```

begin{2D Recursive Pyramid Algorithm}
  for(row=1 to N)
    r_2D_DWT(row, 1);
end{2D Recursive Pyramid Algorithm}

r_2D_DWT(row, j)
begin{r_2d_DWT}
  if(row is odd)
    if(row = 1)
      Lj[1]=filter_row(j-1, 1);
    else
      k=(row-1)/2;
      Lj[row-1]=filter_row(j-1, row-1);
      Lj[row] = filter_row(j-1, row);
      LLj[k] = filter_col(j, row);
    else
      if(row-floor(Nw/2) > 0)
        r_2d_DWT(row-floor(Nw/2)/2, j+1);
      else
        ;
    end{r_2d_DWT}
end{r_2d_DWT}

```

Fig. 9. 2-D-RPA.

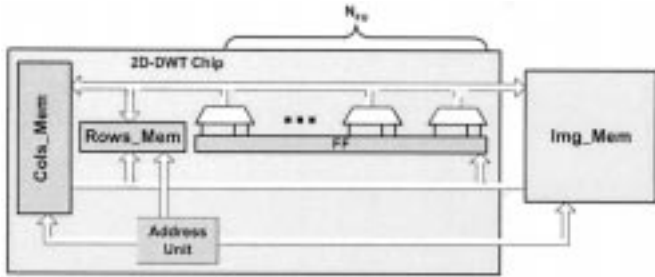


Fig. 10. Line-based architecture.

V. ARCHITECTURE II: LINE-BASED

The line-based architecture scan input image in a row-by-row manner and comply to the following concept: *Proceed to the next layer filtering ASAP, without interleaving filtering along a row.* The line-based architecture is based on an algorithm analogous to the 1-D-RPA algorithm [3]. Although extensions of the RPA algorithm to the 2-D-DWT are referenced by some researchers (e.g., [5], [6]), such an algorithm—to the best of authors' knowledge—has not been presented yet. However, architectures based on the above concept have been described in [14] and [15], the latter of which is also proposed by the JPEG 2000 committee. In this paper, a recursive algorithm for the 2-D-DWT is proposed. We call this algorithm 2-D RPA . The 2-D-RPA is illustrated in Fig. 9 and is the base of the line-based architecture (see Fig. 10) described here.

A direct consequence of interchanging layers' filtering ASAP is that latency is minimized. Additionally, the 2-D-RPA enables the usage of small local memories for the reused data, which are the coefficients of layers L_j and H_j for $j = 1, 2, \dots, L$, and LL_j for $j = 1, 2, \dots, L - 1$ (see Fig. 2). This feature is highly favorable in many cases, since localizing memory accesses can result in lower energy consumption and higher throughput, at the cost, of course, of higher integration area.

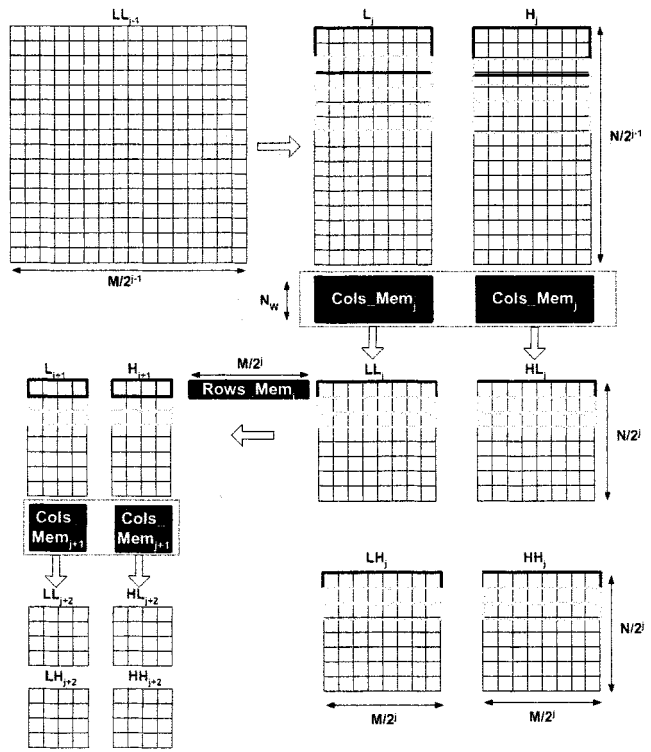


Fig. 11. Local memories with the line-based architecture.

A. Local Memories

To identify how local memories can be used under the line-based architecture, consider the example of Fig. 11, for which a filter of length $N_W = 3$ is used. Filtering the first row of layer LL_{j-1} produces the first row of layers L_j and H_j . The production of layers' LL_j and LH_j (HH_j and HL_j) coefficients is performed by filtering along columns of layer L_j (H_j). To proceed ASAP to the next-layer filtering, N_W rows of LL_{j-1} are filtered; then one filtering along each column of the produced coefficients of layers L_j and H_j must be performed to produce the first row of coefficients at layers LL_j , LH_j , HH_j , and HL_j . Thus, the storage requirements for interchange between L_j and H_j coefficients production, to LL_j , LH_j , HH_j , and HL_j coefficients production is just $N_W \times (M/2^j + M/2^j)$ coefficients. These storage requirements are satisfied by a local memory, here called *Cols_M*. The size of the *Cols_M* is

$$\begin{aligned}
 \text{Cols_M_Size} &= \sum_{j=1}^L \text{Cols_M_Size}_j \\
 &= 2N_W \cdot M \sum_{j=1}^L \frac{1}{2^j} = 2N_W \cdot M \left(1 - \frac{1}{2^L}\right).
 \end{aligned} \tag{18}$$

Remember that L_{j+1} and H_{j+1} layers' coefficients are produced by filtering along rows of LL_j layer and that LL_j is produced in a row-by-row manner. Hence, the storage requirements to interchange between LL_j coefficients production and L_{j+1} and H_{j+1} coefficients productions is equal to just one row of

coefficients of layer LL_j . These storage requirements are satisfied by a local memory, here called Rows_M . The size of the Rows_M is

$$\begin{aligned} \text{Rows}_M\text{-Size} &= \sum_{j=1}^{L-1} \text{Rows}_M\text{-Size}_j \\ &= \sum_{j=1}^{L-1} \frac{M}{2^j} = M \left(1 - \frac{2}{2^L}\right). \end{aligned} \quad (19)$$

With the level-by-level architecture, the storage requirements for interchanging between L_j and H_j coefficients production to LL_j , LH_j , HH_j , and HL_j coefficients production is $(N/2^{j-1})^2$ coefficients. This means that for $j = 1$, the required intermediate storage is equal to the initial image size. This large size of inter-layer buffering is the reason of not considering the usage of local memories in the analysis of the level-by-level architecture.

B. Number of Memory Accesses

Since each input coefficient is read and written only once, and the number of ports of all memories is $N_{\text{FU}} \cdot p$, the number of read and write accesses to Img_M is

$$N_{\text{Reads}}_{\text{Img}_M} = N_{\text{Writes}}_{\text{Img}_M} = \frac{N \cdot M}{N_{\text{FU}} \cdot p}. \quad (20)$$

Cols_M stores the L_j , H_j coefficients, which are intermediate results of the 2-D-DWT. These coefficients are produced (written to Cols_M) during horizontal filtering. Each L_j (H_j) coefficient is written once to Cols_M . Thus, the number of write accesses to Cols_M equals the total number of L_j and H_j coefficients divided by the number of coefficients written during each access

$$\begin{aligned} N_{\text{Writes}}_{\text{Cols}_M} &= \frac{\sum_{j=0}^{L-1} \left(\frac{N}{2^j} \frac{M}{2^j}\right)}{N_{\text{FU}} \cdot p} \\ &= \frac{4N \cdot M}{3N_{\text{FU}} \cdot p} \left(1 - \frac{1}{4^L}\right). \end{aligned} \quad (21)$$

All L_j (H_j) coefficients are consumed (read from Cols_M) during vertical filtering. For level j , the filtering of all L_j (or H_j) coefficients that are stored in Cols_M results in only one pair of rows of $LL_j - LH_j$ (or $HL_j - HH_j$) coefficients. In other words, to produce a pair of rows of $LL_j - LH_j$ (or $HL_j - HH_j$) coefficients, $2N_W \cdot M/2^j$ L_j (or H_j) coefficients must be read. Since at level j the total number of pairs of rows of $LL_j - LH_j$ (or $HL_j - HH_j$) coefficients is $N/2^j$, the total number of accesses to Cols_M can be found as follows:

$$\begin{aligned} N_{\text{Reads}}_{\text{Cols}_M} &= \frac{2N_W \sum_{j=0}^{L-1} \frac{N}{2^j} \sum_{j=0}^{L-1} \frac{M}{2^j}}{N_{\text{FU}} \cdot p} \\ &= N_W \frac{2N \cdot M}{3N_{\text{FU}} \cdot p} \left(1 - \frac{1}{4^L}\right). \end{aligned} \quad (22)$$

To produce one row of coefficients at layer L_{j+1} and H_{j+1} , $M/2^j$ coefficients must be read from Rows_M . Thus, the number of coefficients read from Rows_M is equal to the

product of $M/2^j$ times the number of rows at layer L_{j+1} , which is $N/2^j$. Hence, the total number read accesses from (write accesses to) Rows_M is

$$\begin{aligned} N_{\text{Reads}}_{\text{Rows}_M} &= N_{\text{Writes}}_{\text{Rows}_M} \\ &= N \cdot M \frac{\sum_{j=1}^{L-1} \left(\frac{1}{2^j}\right)^2}{N_{\text{FU}} \cdot p} \\ &= \frac{N \cdot M}{3N_{\text{FU}} \cdot p} \left(1 - \frac{4}{4^L}\right). \end{aligned} \quad (23)$$

C. Throughput

In the following analysis, we use the symbols t_{Img_M} , t_{Cols_M} , and t_{Rows_M} to indicate the number of clock cycles per Img_M , Cols_M , and Rows_M , respectively. Thus, assuming a 100% utilization of all filters in the linear array, the number of clock cycles to perform the 2-D-DWT with line-based architecture is

$$\begin{aligned} N_{\text{Cycles}} &= t_{\text{Img}_M} \cdot N_{\text{Reads}}_{\text{Img}_M} + t_{\text{Cols}_M} \\ &\quad \cdot N_{\text{Reads}}_{\text{Cols}_M} + t_{\text{Rows}_M} \cdot N_{\text{Reads}}_{\text{Rows}_M} \\ &= \frac{N \cdot M}{N_{\text{FU}} \cdot p} \left[t_{\text{Img}_M} + \frac{2N_W \cdot t_{\text{Cols}_M}}{3} \left(1 - \frac{1}{4^L}\right) \right. \\ &\quad \left. + \frac{t_{\text{Rows}_M}}{3} \left(1 - \frac{4}{4^L}\right) \right]. \end{aligned} \quad (24)$$

Hence, throughput in terms of input coefficients processed per time unit is

$$\begin{aligned} \text{Throughput} &= f_{\text{clk}} \left(\frac{1}{N_{\text{FU}} \cdot p} \left[t_{\text{Img}_M} + \frac{2N_W t_{\text{Cols}_M}}{3} \right. \right. \\ &\quad \left. \left. \cdot \left(1 - \frac{1}{4^L}\right) + \frac{t_{\text{Rows}_M}}{3} \left(1 - \frac{4}{4^L}\right) \right] \right)^{-1}. \end{aligned} \quad (25)$$

D. Energy

To derive a formula for the energy dissipation of the line-based architecture, we first consider the typical case according to which: 1) an input image is stored off-chip and 2) local memories (Cols_M and Rows_M) are stored on-chip. This allocation of memory blocks is considered typical mainly due to the memory blocks' size. For example, in the extreme case that $N = M = 1024$, $N_W = 9$, and $L = 6$, the sum of sizes of Cols_M and Rows_M is less than 20 K coefficients, which modern technologies allow to be stored on-chip. On the other hand, Img_M size is 64 K coefficients, even when $N = M = 256$. Under this allocation, energy dissipation is

$$\begin{aligned} \text{Energy} &= N_{\text{Accesses}}_{\text{Img}_M} \\ &\quad \cdot [d \cdot N_{\text{FU}} \cdot p \cdot 10^{-11} \cdot V_{DD}^2 \\ &\quad + f(\text{Img}_M\text{-Size}, d, N_{\text{FU}} \cdot p)] \\ &\quad + N_{\text{Accesses}}_{\text{Cols}_M} \\ &\quad \cdot f(\text{Cols}_M\text{-Size}, d, N_{\text{FU}} \cdot p) \\ &\quad + N_{\text{Accesses}}_{\text{Rows}_M} \\ &\quad \cdot f(\text{Rows}_M\text{-Size}, d, N_{\text{FU}} \cdot p) \end{aligned} \quad (26)$$

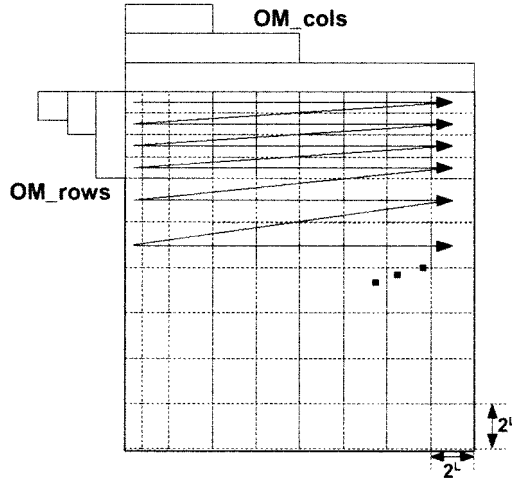


Fig. 12. Input image scan with the block-based approach.

where

$$N_{\text{Accesses}_{\text{mem_block}}} = N_{\text{Reads}_{\text{mem_block}}} + N_{\text{Writes}_{\text{mem_block}}}.$$

From (26) and (18)–(23), energy dissipation can be expressed in terms of the basic parameters used throughout this paper.

Although current technology does not allow for on-chip integration of Img_M , it is expected that in the near future Img_M will migrate on-chip, at least for small image sizes. To estimate the energy dissipation of the line-based architecture in such a case, the term $d \cdot N_{\text{FU}} \cdot p(40 \cdot 10^{-12}/4) \cdot V_{\text{DD}}^2$ should be removed from (26).

VI. ARCHITECTURE III: BLOCK-BASED

The block-based approach proposed in [6] is based on the following concept: *Produce the next parent-children-tree (pct) ASAP*. The root of a pct is a quadruplet of coefficients at the highest level of decomposition (i.e., LL_L, LH_L, HH_L, LH_L). For the production of each new pct, an input image's block with size equal to $2^L \times 2^L$ is required. The input image's (nonoverlapping) blocks are traversed in the order indicated by the directed lines, as in Fig. 12.³ This block-by-block traversal is a disadvantage for the block-based architecture, since this short-of-input image scanning is unsuitable for streaming applications and, furthermore, it requires a rather complex addressing.

A block diagram of the block-based hardware architecture is illustrated in Fig. 13. The operation of the block-based architecture is as follows. A $2^L \cdot 2^L$ block is fetched from the Img_M to a local (possibly on-chip) memory called inter-pass memory and denoted as IPM. As mentioned earlier, the size of the block fetched to IPM is such that allows for the production of a quadruplet of coefficients at the highest level of decomposition. Filtering is performed within the limits of the block and the output coefficients are written back to the IPM, exactly in the

³The blocks near the top and left limits of the image have different sizes to satisfy initialization needs (further details can be found in [6]).

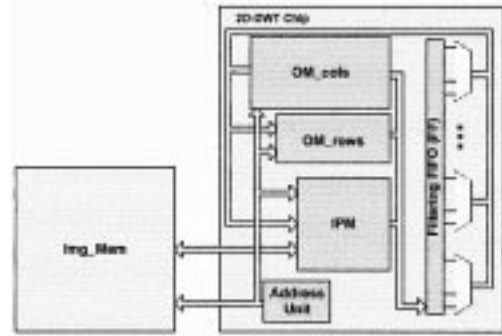


Fig. 13. Overlapping.

same way as in the level-by-level architecture (see Section IV). Then the decomposition of the block is written back to Img_M and the next block is fetched in the IPM, and so on.

Since an image is filtered in a block-by-block manner, filtering along a row of the initial image is interleaved several times. The same holds for all the decomposition layers too. Recall that two successive filtering operations along the same column or row share $N_W - 2$ coefficients. This fact imposes the existence of two extra local memories to store the overlapped coefficients. To clear this out, let us consider a reference block. In Fig. 14, the gray rectangles indicate the reference block decomposition at layers LL_{j-1}, L_j , and H_j . The green and yellow rectangles indicate the adjacent blocks decomposition layers in the horizontal and vertical direction, respectively. To perform filtering within the green block at layer LL_{j-1} , we need $N_W - 2$ coefficients produced during decomposition of the gray block at the same layer. In a manner similar to perform filtering within the yellow block at layer L_{j-1} , we need $N_W - 2$ coefficients produced during decomposition of the gray block at the same layer. To avoid the recalculation of the overlapped coefficients, which would introduce a speed penalty and increase the size of IPM, two extra memory blocks are used with the block-based architecture: the first one stores coefficients of input and of layers LL_j ($j = 1, 2, \dots, L-1$), and is named overlap memory along rows (OM.Rows), while the second stores coefficients of layers L_j and H_j ($j = 1, 2, \dots, L$), and is named overlap memory along columns (OM.Cols).

A slightly different hardware architecture results, if instead of fetching one block at the time, a super-block of $N_Y \times N_X$ blocks is fetched in the IPM. Filtering within such a super-block enables the production of $N_Y \times N_X$ pct's without accessing the Img_M . The differences among initial description and this variation of the block-based architecture are just in terms of memory sizes and number of accesses to each memory. For this reason, from this point forward we will study both in a unified way. Specifically, we will refer to both alternatives using the name *block based architecture/approach* and the oncoming analysis will consider N_X and N_Y as two extra implementation parameters.

The pseudo-code of Fig. 15 provides a more compact description of the basic steps of the data and control flow of the block-based architecture. In Fig. 15, initialization and finalization phenomena are omitted for the sake of simplicity.

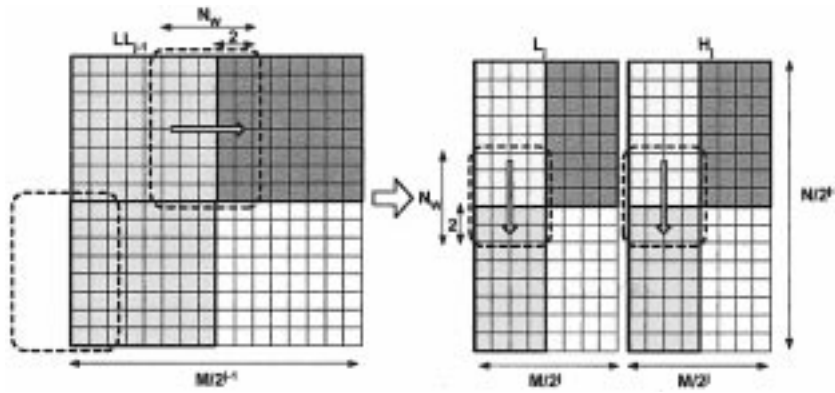


Fig. 14. Block-based architecture.

```

begin{Block-Based Algorithm}
for( x=0 to N/(NX2L) )
  for(y=0 to N/(NY2L))
    fetch a block from Img_Mem[x (NX2L)] [y (NY2L)] and store it to IPM;
    for(j=0 to L)
      for(row=0 to NX2L/2j)
        fetch Nw coefficients from OM_Rows[j][row] and store them to FF;
        filtering along row at level j;
        /*output coeffs are stored in IPM, in-place of input coeffs*/
        store last Nw coefficients from FF and store them to OM_Rows[rows];
      for(col=0 to NY2L/2j)
        fetch Nw coefficients from OM_Cols[j][col] and store them to FF;
        filter along col at level j;
        /*output coeffs are stored in IPM, in-place of input coeffs*/
        store last Nw coefficients from FF and store them to OM_Rows[rows];
      output IPM to Img_Mem[x (NX2L)] [y (NY2L)];
    end{Block-Based Algorithm}

```

Fig. 15. Algorithm for the block-based architecture.

A. Local Memories

As previously mentioned, the IPM can store $N_X \cdot N_Y$ blocks with size $2^L \cdot 2^L$. Thus

$$\text{IPM.Size} = N_X \cdot N_Y \cdot 4^L. \quad (27)$$

An input image is scanned in a row of (super-)blocks-by-row of (super-)blocks order. For each block, filtering along all layers of decomposition is performed. This traversal imposes OM_Cols to store $N_W - 2$ coefficients for each column and each decomposition level. The number of columns at decomposition level j is $M/2^j$. Hence, the size of OM_Cols is

$$\begin{aligned} \text{OM.Cols.Size} &= (N_W - 2) \sum_{j=0}^{L-1} \frac{M}{2^j} \\ &= 2M(N_W - 2) \left(1 - \frac{1}{2^L}\right). \end{aligned} \quad (28)$$

The traversal of the block-by-block architecture imposes to store in the OM_Rows, $N_W - 2$ coefficients for each row and

each decomposition level of a (super-)block. The number of rows at decomposition level j of a (super-)block is $N_Y \cdot 2^L/2^j$. Hence, the size of OM_Rows is

$$\begin{aligned} \text{OM.Rows.Size} &= (N_W - 2) \sum_{j=0}^{L-1} \frac{N_Y \cdot 2^L}{2^j} \\ &= 2N_Y(N_W - 2)(2^L - 1). \end{aligned} \quad (29)$$

B. Memory Accesses

Since each input coefficient is read and written only once, and the number of ports of all memories is $N_{FU} \cdot p$, the number of read and write accesses to *Img_M* is

$$N_{\text{Reads}}_{\text{Img}_M} = N_{\text{Writes}}_{\text{Img}_M} = \frac{N \cdot M}{N_{FU} \cdot p}. \quad (30)$$

Since, for each (super-)block, the 2-D-DWT is performed in the same way as in the level-by-level approach, we can use (13) to compute the number of coefficients read from IPM per block.

Of course, $\text{Img_}M$ in (13) must be replaced with IPM. Additionally, each block is written back to $\text{Img_}M$. The total number of (super-)blocks is $N \cdot M / (N_X \cdot N_Y \cdot 4^L)$. Hence

$$\begin{aligned} N_{\text{ReadsIPM}} &= N_{\text{WritesIPM}} \\ &= \frac{N \cdot M}{N_X \cdot N_Y \cdot 4^L} \\ &\quad \cdot \left(\frac{8N_X \cdot N_Y \cdot 4^L}{3N_{\text{FU}} \cdot p} \cdot \left(1 - \frac{1}{4^L}\right) + N_X \cdot N_Y \cdot 4^L \right) \\ &= \frac{8N \cdot M}{3N_{\text{FU}} \cdot p} \cdot \left(1 - \frac{1}{4^L}\right) + N \cdot M. \end{aligned} \quad (31)$$

$N_W - 2$ coefficients are fetched/stored from/to OM_Cols for each column of each decomposition level of each (super-)block. The number of (super-)blocks is $N \cdot M / (N_X \cdot N_Y \cdot 4^L)$; the number of column at decomposition level j of a block is $N_Y \cdot 2^L / 2^j$. Thus

$$\begin{aligned} N_{\text{ReadsOM_Rows}} &= N_{\text{WritesOM_Rows}} \\ &= \frac{N_W - 2}{N_{\text{FU}} \cdot p} \cdot \frac{N \cdot M}{N_X \cdot N_Y \cdot 4^L} \cdot N_Y \cdot 2^L \cdot \sum_{j=0}^{L-1} \frac{1}{2^j} \\ &= \frac{2N \cdot M \cdot N_W - 2}{N_{\text{FU}} \cdot p \cdot N_X} \cdot \left(\frac{1}{2^L} - \frac{1}{4^L} \right). \end{aligned} \quad (32)$$

$N_W - 2$ coefficients are fetched/stored from/to OM_Rows for each row of each decomposition level of each (super-)block. Thus in the same way as in the case of OM_Cols , the number of read (write) accesses to OM_Rows is

$$\begin{aligned} N_{\text{ReadsOM_Cols}} &= N_{\text{WritesOM_Cols}} \\ &= \frac{2N \cdot M \cdot (N_W - 2)}{N_{\text{FU}} \cdot p \cdot N_Y} \cdot \left(\frac{1}{2^L} - \frac{1}{4^L} \right). \end{aligned} \quad (33)$$

C. Throughput

In the following analysis, we use the symbols $t_{\text{Img_}M}$, t_{IPM} , $t_{\text{OM_Cols}}$, and $t_{\text{OM_Rows}}$ to indicate the number of clock cycles per $\text{Img_}M$, IPM, OM_Cols , and OM_Rows , respectively. Thus, assuming a 100% utilization of all filters in the linear array, the number of clock cycles to perform the 2-D-DWT with block-based architecture is

$$\begin{aligned} N_{\text{Cycles}} &= N_{\text{CyclesReads}} = t_{\text{Img_}M} \cdot N_{\text{ReadsImg_}M} \\ &\quad + t_{\text{IPM}} \cdot N_{\text{ReadsIPM}} \\ &\quad + t_{\text{OM_Rows}} \cdot N_{\text{ReadsOM_Rows}} \\ &\quad + t_{\text{OM_Cols}} \cdot N_{\text{ReadsOM_Cols}} \\ &= \frac{N \cdot M}{N_{\text{FU}} \cdot p} \left[t_{\text{Img_}M} + \frac{8t_{\text{IPM}}}{3} \left(1 - \frac{1}{4^L}\right) \right. \\ &\quad \left. + 2(N_W - 2) \cdot \left(\frac{1}{2^L} - \frac{1}{4^L} \right) \right. \\ &\quad \left. \cdot \left(\frac{t_{\text{OM_Rows}}}{N_Y} + \frac{t_{\text{OM_Cols}}}{N_X} \right) \right]. \end{aligned} \quad (34)$$

TABLE II
COMPARISON PARAMETERS

Par.	Typical Values	Description
N	$2^n, n = 7, 8, \dots$	Input image dimension
M	$2^m, m = 7, 8, \dots$	Input image dimension
d	8, 9, ..., 32	Number of bits per coefficient
L	1, 2, ..., 7	2D-DWT levels of decomposition
N_W	2, 3, ..., 15	max. filter width
p	1, 2	1: Parallel filter, 2: Parallel, throughput-optimized filter
N_{FU}	1, 2, 3, 4	Number of filters in a linear array
t_m	1, 2, ..., 20	Access latency for memory m
f_{clk}	10 - 400 MHz	Clock Frequency

Hence, throughput in terms of input coefficients processed per time unit is

$$\begin{aligned} \text{Throughput} &= f_{\text{clk}} \left(\frac{1}{N_{\text{FU}} \cdot p} \left[t_{\text{Img_}M} + \frac{8t_{\text{IPM}}}{3} \left(1 - \frac{1}{4^L}\right) \right. \right. \\ &\quad \left. \left. + 2 \cdot (N_W - 2) \cdot \left(\frac{1}{2^L} - \frac{1}{4^L} \right) \right. \right. \\ &\quad \left. \left. \cdot \left(\frac{t_{\text{OM_Rows}}}{N_Y} + \frac{t_{\text{OM_Cols}}}{N_X} \right) \right] \right)^{-1}. \end{aligned} \quad (35)$$

D. Energy

To come up with a formula for the energy dissipation of the block-based architecture, we must first allocate each memory block on- or off-chip. IPM, OM_Rows , and OM_Cols are typically stored on-chip, since the sum of their sizes is less than 32 K coefficients, even for the extreme case that $N = M = 1024$, $L = 6$, $N_W = 9$, and $N_X = N_Y = 2$. For $\text{Img_}M$, we consider two cases. According to the first case, $\text{Img_}M$ lie off-chip, while for the second case, $\text{Img_}M$ lie on-chip. In the first case, the energy dissipation of the block-based architecture is estimated by the following equation:

$$\begin{aligned} \text{Energy} &= N_{\text{ReadsImg_}M} \\ &\quad \cdot [d \cdot N_{\text{FU}} \cdot p \cdot 10^{-11} \cdot V_{DD}^2 \\ &\quad \quad + f(\text{Img_}M_{\text{Size}}, d, N_{\text{FU}} \cdot p)] \\ &\quad + N_{\text{ReadsIPM}} \cdot f(\text{IPM}_{\text{Size}}, d, N_{\text{FU}} \cdot p) \\ &\quad + N_{\text{ReadsOM_Rows}} \cdot f(\text{OM_Rows}_{\text{Size}}, d, N_{\text{FU}} \cdot p) \\ &\quad + N_{\text{ReadsOM_Cols}} \cdot f(\text{OM_Cols}_{\text{Size}}, d, N_{\text{FU}} \cdot p). \end{aligned} \quad (36)$$

In the case that $\text{Img_}M$ is stored on-chip the term $d \cdot N_{\text{FU}} \cdot p (40 \cdot 10^{-12} / 4) \cdot V_{DD}^2$ should be removed from (36). From (27)–(33) and (36), energy dissipation can be expressed in terms of the basic parameters used throughout this paper.

VII. COMPARISONS

In this section, we attempt a comparison in terms of memory requirements, throughput, and energy dissipation of the design alternatives of the 2-D-DWT presented in Sections IV–VI. The comparison is based on parametric (13)–(36). Table II

TABLE III
THE PARAMETRIC EQUATIONS FOR MEMORY SIZE, NUMBER OF ACCESSES, THROUGHPUT AND ENERGY DISSIPATION

	Memory Size	Number of Accesses
Level-by-Level		
<i>Img_M</i>	NM	$\frac{16}{3} \left(1 - \frac{1}{4L}\right) \frac{N \cdot M}{N_{FU} \cdot p}$
Line-Based		
<i>Img_M</i>	NM	NM
<i>Cols_M</i>	$2N_W \cdot M \left(1 - \frac{1}{2L}\right)$	$(N_W + 2) \frac{2N \cdot M}{3N_{FU} \cdot p} \left(1 - \frac{1}{4L}\right)$
<i>Rows_M</i>	$M \left(1 - \frac{2}{2L}\right)$	$\frac{2N \cdot M}{3N_{FU} \cdot p} \left(1 - \frac{4}{4L}\right)$
Block-Based		
<i>Img_M</i>	NM	NM
<i>IPM</i>	$N_X \cdot N_Y \cdot 4^L$	$\frac{16N \cdot M}{3N_{FU} \cdot p} \cdot \left(1 - \frac{1}{4L}\right) + N \cdot M$
<i>OM_Cols</i>	$2M(N_W - 2) \left(1 - \frac{1}{2L}\right)$	$\frac{4N \cdot M \cdot (N_W - 2)}{N_{FU} \cdot p \cdot N_X} \left(\frac{1}{2L} - \frac{1}{4L}\right)$
<i>OM_Rows</i>	$2N_Y(N_W - 2)(2^L - 1)$	$\frac{4N \cdot M \cdot (N_W - 2)}{N_{FU} \cdot p \cdot N_Y} \left(\frac{1}{2L} - \frac{1}{4L}\right)$
Throughput		
Level-by-Level	$f_{clk} \cdot \left(\frac{8}{3} \left(1 - \frac{1}{4L}\right) \frac{t_{Img_M}}{N_{FU} \cdot p}\right)^{-1}$	
Line-Based	$f_{clk} \cdot \left(\frac{1}{N_{FU} \cdot p} \left[t_{Img_M} + \frac{(N_W + 2)t_{Cols_M}}{3} \left(1 - \frac{1}{4L}\right) + \frac{t_{Rows_M}}{3} \left(1 - \frac{4}{4L}\right) \right]\right)^{-1}$	
Block-Based	$f_{clk} \cdot \left(\frac{1}{N_{FU} \cdot p} \left[t_{Img_M} + \frac{8t_{IPM}}{3} \left(1 - \frac{1}{4L}\right) + 2(N_W - 2) \left(\frac{1}{2L} - \frac{1}{4L}\right) \left(\frac{t_{OM_Rows}}{N_Y} + \frac{t_{OM_Cols}}{N_X}\right) \right]\right)^{-1}$	
Energy		
Level-by-Level		
<i>Img_M</i> *	$\frac{16}{3} \left(1 - \frac{1}{4L}\right) \frac{N \cdot M}{N_{FU} \cdot p} \cdot \left d \cdot N_{FU} \cdot p \frac{40 \cdot 10^{-12}}{4} \cdot V_{DD}^2 + f(N \cdot M, d, N_{FU} \cdot p) \right $	
Line-Based		
<i>Img_M</i> *	$\frac{N \cdot M}{N_{FU} \cdot p} \cdot \left d \cdot N_{FU} \cdot p \frac{40 \cdot 10^{-12}}{4} \cdot V_{DD}^2 + f(N \cdot M, d, N_{FU} \cdot p) \right $	
<i>Cols_M</i>	$(N_W + 2) \frac{2N \cdot M}{3N_{FU} \cdot p} \left(1 - \frac{1}{4L}\right) \cdot f\left(2N_W \cdot M \left(1 - \frac{1}{2L}\right), d, N_{FU} \cdot p\right)$	
<i>Rows_M</i>	$\frac{2N \cdot M}{3N_{FU} \cdot p} \left(1 - \frac{4}{4L}\right) \cdot f\left(M \left(1 - \frac{2}{2L}\right), d, N_{FU} \cdot p\right)$	
Block-Based		
<i>Img_M</i> *	$\frac{16}{3} \left(1 - \frac{1}{4L}\right) \frac{N \cdot M}{N_{FU} \cdot p} \cdot \left d \cdot N_{FU} \cdot p \frac{40 \cdot 10^{-12}}{4} \cdot V_{DD}^2 + f(N \cdot M, d, N_{FU} \cdot p) \right $	
<i>IPM</i>	$\left[\frac{16N \cdot M}{3N_{FU} \cdot p} \cdot \left(1 - \frac{1}{4L}\right) + N \cdot M \right] \cdot f\left(N_X \cdot N_Y \cdot 4^L, d, N_{FU} \cdot p\right)$	
<i>OM_Cols</i>	$\frac{4N \cdot M \cdot (N_W - 2)}{N_{FU} \cdot p \cdot N_X} \left(\frac{1}{2L} - \frac{1}{4L}\right) f\left(2M(N_W - 2) \left(1 - \frac{1}{2L}\right), d, N_{FU} \cdot p\right)$	
<i>OM_Rows</i>	$\frac{4N \cdot M \cdot (N_W - 2)}{N_{FU} \cdot p \cdot N_Y} \left(\frac{1}{2L} - \frac{1}{4L}\right) f\left(2N_Y(N_W - 2)(2^L - 1), d, N_{FU} \cdot p\right)$	

* if *Img_M* is stored on-chip then the term $d \cdot N_{FU} \cdot p \frac{40 \cdot 10^{-12}}{4} \cdot V_{DD}^2$ must be removed.

summarizes the performed analysis parameters, while Table III summarizes the parametric equations. Although analytical equations are available, the identification of the conditions under which the one architecture overcomes the others in a purely analytical fashion, however, is almost infeasible. For this reason, we perform the comparison for typical cases of the 2-D-DWT, and draw general conclusions whenever possible. We consider as typical examples some of the 2-D-DWT included in the JPEG2000 final committee draft [16], namely the 2-D-DWT based on the 5/3, 9/7, and 10/18 filters.

A. Memory Requirements

The level-by-level approach has smaller memory requirements than the line-based and block-based architectures. Furthermore, the memory requirements of level-by-level approach are not modified with input dimensions, nor type of filter, nor number of decomposition levels.

The only memory block (*Img_M*) that is required by the level-by-level is also present in the line- and block-based architectures. Additionally, the line- and block-based architectures succeed to store intermediate results in local memory blocks

that are typically stored on-chip. The sizes of the local memories varies with input image dimensions, number of decomposition layers, and filter width. Fig. 16 illustrates the sizes of the local memories with the line- and block-based architectures for 5/3, 9/7, and 10/18 2-D-DWT. As can be observed, the block-based has 15%–45% smaller requirements for local storage than the line-based architecture in the majority of cases. This is not true in cases where the number of decomposition levels is greater than six (e.g., 7, 2048, or 6, 512), where the the block-based has 25%–100% greater requirements for local storage than the line-based architecture.

Furthermore, an interesting observation is that the number of decomposition levels slightly affects memory sizes for the line-based architecture, while heavily affects memory sizes of the block-based architecture. For example in the case of 1024×1024 input image and 5/3 2-D-DWT, the line-based architecture requires local storage of 9.5 Kcoeffs for three levels and 11 Kcoeffs for seven levels of decomposition, while the block-based architecture requires local storage of 5.3 and 22.7 Kcoeffs, respectively. This must be taken into account, especially in cases that the design goal is a DWT engine programmable in terms of decomposition levels.

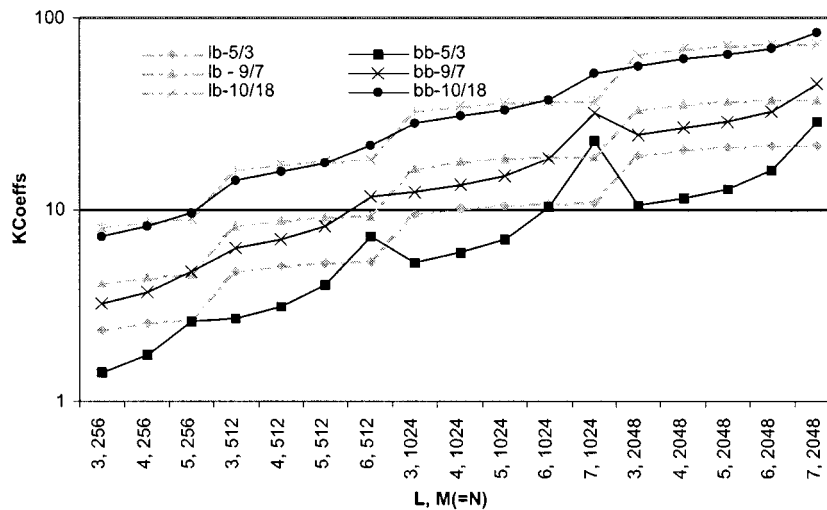


Fig. 16. Local-memories' size for the line- and block-based architectures.

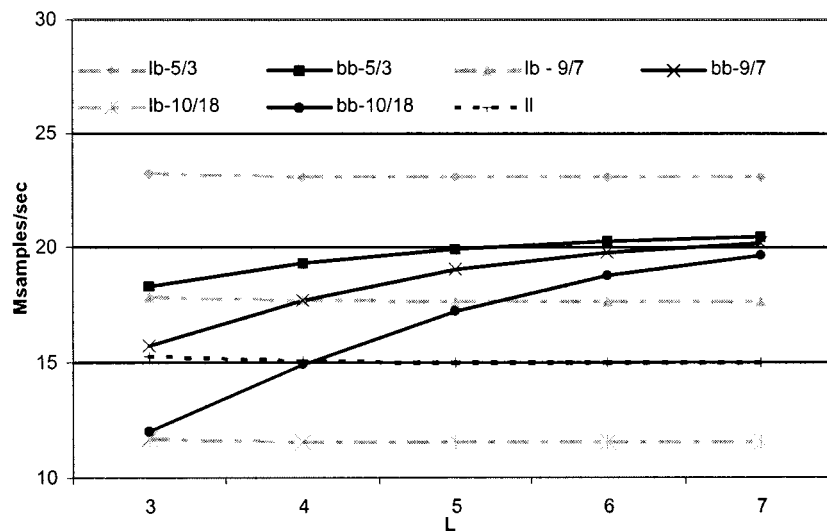


Fig. 17. Throughput for $t_{\text{Img}_M} = 5t_{\text{on-chip}} = 5$.

Summarizing the above results to:

- the level-by-level architecture do not use any local memories and thus has the smaller memory requirements among the three architectures;
- the block-based architecture requires less local memory storage than the line-based architecture, when the number of decomposition levels is in the range of 1–6; if the number of decomposition levels is greater than 6 then the situation is reversed;
- the greater the filter length is, the smaller the differences in terms of memory requirements for the block-based and the line-based architecture becomes;
- the number of decomposition levels do not, slightly and significantly affect the memory requirements of the level-by-level, line-based and block-based architectures, respectively.

B. Throughput

To perform a comparison among the alternative architectures in terms of throughput, we consider a typical case according

to which: 1) the clock frequency is 100 MHz and 2) all memories are dual-ported ($N_{\text{FU}} \cdot p = 2$). Furthermore, two cases for the relation among the latency of Img_M and the latency of the local memories (IPM, OM_Cols, OM_Rows, Cols_M, and Rows_M) are studied. The first case, according to which $t_{\text{Img}_M} = 5t_{\text{local}_m} = 5$, is representative of the case that Img_M is stored off-chip and all local-memories are stored on-chip, while the second, according to which $t_{\text{Img}_M} = t_{\text{local}_m} = 1$, is representative of the case that both Img_M and local-memories are stored on-chip. It is noted, that the following comparison ignores image dimensions (N, M), since throughput in terms of Msamples/s does not depend on them (see Table III).

Fig. 17 illustrates the throughput in terms of Msamples/s of all three architectures for 5/3, 9/7, and 10/18 2-D-DWT, when $t_{\text{Img}_M} = 5t_{\text{local}_m} = 5$. In the case of the 5/3 2-D-DWT, the line-based is 11%–21% faster than block-based architecture. Additionally, the line-based architecture overturns the block-based architecture by 12% in the case of the 9/7 2-D-DWT for three decomposition levels. On the other hand, for decompo-

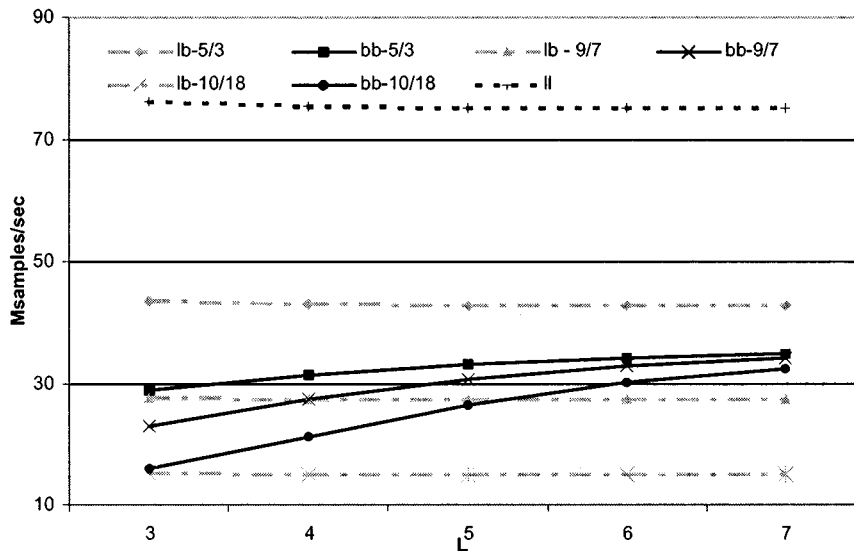


Fig. 18. Throughput for $t_{\text{Img}_M} = t_{\text{on-chip}} = 1$.

sition levels from 4 up to 7 in the case of the 9/7 2-D-DWT, and for the whole range of decomposition levels in the case of the 10/18 2-D-DWT, the block-based is 3%–70% faster than the line-based architecture. A more detailed analysis of (25) and (35) results in the conclusion that the line-based is faster than the block-based architecture either if $N_W \leq 7$, or if $(7 < N_W \leq 15) \wedge (L \leq 3)$. Obviously, in any other case, the situation is reversed. Finally, as a consequence of Img_M 's latency, for all filters the level-by-level is slower than the other two architectures.

Fig. 18 illustrates the throughput in terms of Msamples/s of all three architectures for 5/3, 9/7, and 10/18 2-D-DWT, when $t_{\text{Img}_M} = t_{\text{local_mems}}$. In this case, the parameter that defines the comparison outcome is the total number of memory accesses, since there is no latency difference for accesses to Img_M and local memories. Hence, as expected the level-by-level architecture, which performs the smallest possible number of memory accesses, is the one that is faster than the other two architectures for all filter sets and number of decomposition levels. The comparison among the other two architectures leads to the same conclusions as in the case that $t_{\text{Img}_M} = 5t_{\text{local_mems}}$, but now the percentage differences are greater from 5% to 17%.

Summarizing the above leads to the following general statements.

- When Img_M is stored off-chip and local memories are stored on-chip, the faster among the three architectures is either: 1) the line-based in the case of small filter lengths ($N_W \leq 7$), or in the case of a small number of decomposition levels ($L \leq 3$) and filter lengths in the range: $7 < N_W \leq 15$, or 2) the block-based in any other case.
- When both Img_M and local memories are stored on-chip, the level-by-level is the faster among the three alternative architectures.
- Throughput of level-by-level and line-based architecture negligibly depends on decomposition levels, while the block-based architecture throughput increases with the number of decomposition levels.

C. Energy

One off-chip access is 10 to 100 times more energy consuming than one on-chip access, depending on technology, off-chip buses length, etc. For this reason, the comparison of the three alternative architectures for the 2-D-DWT is performed separately for the cases that Img_M is stored off-chip and on-chip.

For the first case, the level-by-level approach is by far the less energy-efficient architecture, since with this architecture all coefficients are fetched from and stored to the off-chip Img_M . On the other hand, the other two architectures access the intermediate results (i.e., coefficients H_j , L_j and LL_j , $j \neq L$) from the local on-chip memories, and thus succeed to significantly reduce energy requirements. Hence, for this case, the comparison in terms of energy dissipation is performed only among the line- and block-based architectures. Since both these architectures perform the same number of accesses to the off-chip Img_M [(20)=(30)], the comparison among them is focused only in the energy consumed due to accesses to the on-chip local memories. Fig. 19 shows the energy consumed due to on-chip memory accesses for the cases of a 5/3, 9/7, and 10/18 2-D-DWT. From this figure, it can be observed that in all cases, the line-based consumes more energy due to on-chip memory accesses than the block-based architecture. The difference in terms of energy dissipation due to accesses to the local memories among the two architectures is significant and lies in the range of 13%–85%. This difference is due to the fact that the line-based architecture performs the dominant majority of on-chip memory accesses to Cols_M , while the block-based architecture performs the dominant majority of on-chip memory accesses to the smaller IPM , where the energy-cost per access is much lower.

Fig. 20 illustrates the energy dissipation for all three architectures, for the case that both Img_M and local memories lie on-chip. In this case, the factors that determine the outcome of the comparison are the number of memory accesses and the way this accesses are distributed to the memory blocks. Specifically, to succeed the storage of intermediate results in

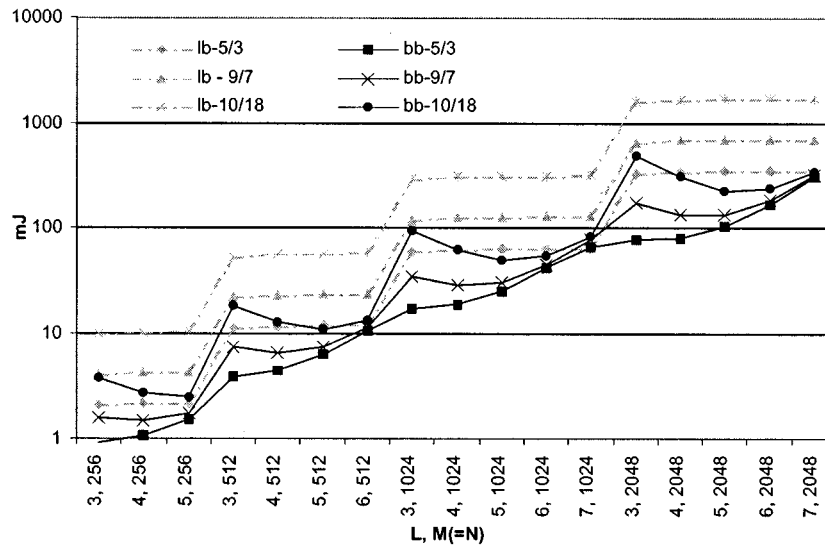


Fig. 19. Energy consumption of local (on-chip) memories with the line- and block-based architectures.

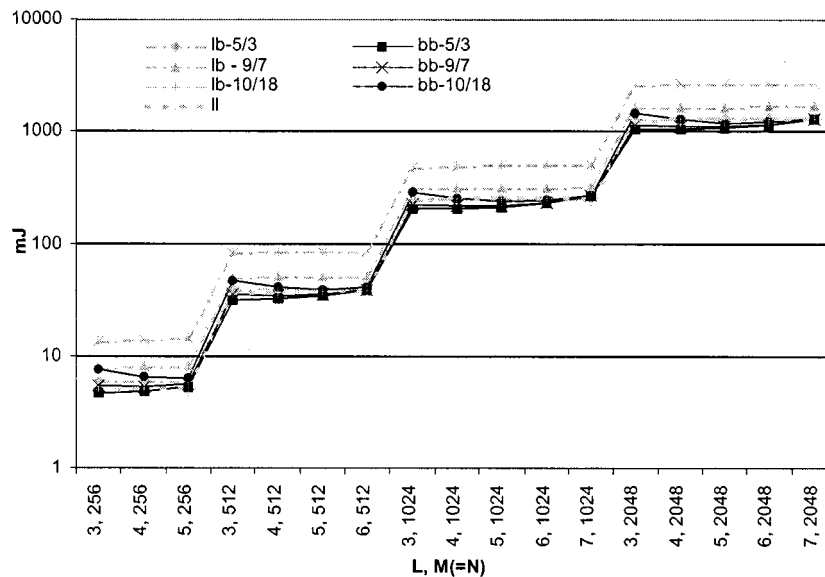


Fig. 20. Energy consumption (Img_M and local-memories: on-chip).

the smaller than Img_M local memories, the block-based and line-based architectures have a higher number of memory accesses when compared to the level-by-level architecture. Hence, from the energy consumption point of view, on the one hand, the level-by-level has the advantage of performing less memory accesses, and on the other hand, the block- and line-based architectures have the advantage of performing the majority of accesses to memories with smaller size than the size of Img_M . As shown in Fig. 20, this tradeoff turns in favor of the block-based architecture for relatively small filter widths and in favor of the level-by-level architecture for relatively large filter widths. This is due to the fact that memory requirements and the number of memory accesses for the level-by-level architecture do not depend on filter width, while for the block- and line-based architectures, as the width of the filter increases, the size of the local memories and the number of accesses to them also increase [(18), (28), (29) and (23), (32) and (33), respectively]. Furthermore, it can be observed that the small image dimensions ($N \times M$) are in favor of the level-by-level

architecture, while the large image dimensions are in favor of the block-based architecture. This is rational since the level-by-level performs all of its accesses to Img_M the size of which increases with image dimensions, while a significant part of the accesses of the block-based architecture is performed on memory blocks (IPM and OM_Rows) the size of which do not depend on image dimensions. A detailed analysis of energy consumption data leads to the conclusion that the most energy efficient among the three architectures studied here is: (1) either the level-by-level when $((N_W > 11) \wedge (N \times M < 2^{19})) \vee ((5 < N_W \leq 11) \wedge (N \times M < 2^{17}))$ or (2) the block-based when: $(N_W \leq 5) \vee (N \times M \geq 2^{19})$. It is noted that when $((5 < N_W < 11) \wedge (2^{17} \leq N \times M < 2^{19}))$, the differences among the block-based and level-by-level architectures in terms of energy dissipation are negligible ($< 3\%$). For example, as shown in Fig. 20, for 5/3 2-D-DWT the level-by-level consumes on average 9% more energy than the block-based, while for the 18/10 2-D-DWT and $N \times M = 256, 512$ the level-by-level consumes on average almost 22% less than the

TABLE IV
COMPARISON SUMMARY

	Level-by-Level	Block-Based	Line-Based	
Storage	Min.	Max. for: $L < 6$	Max. for: $L > 6$	
Throughput	Img_M and Local_Mems: On-Chip			
	High	Low	Low	
	Img_M: Off-Chip and Local_Mems: On-Chip			
	Low	$(N_W > 7) \vee ((7 < N_W \leq 15) \wedge (L < 3))$		High
		Moderate		
$(N_W > 15) \vee ((7 < N_W \leq 15) \wedge (L > 3))$				
	High	Moderate		
Latency	High	Low	Low	
Energy	Img_M and Local_Mems: On-Chip			
	$((N_W > 11) \wedge (N \times M < 2^{19})) \vee ((5 < N_W \leq 11) \wedge (N \times M < 2^{17}))$			
	Low	Moderate	High	
	$(N_W \leq 1) \wedge (N \times M \geq 8^{19})$			
	Moderate	Low		
	Img_M: Off-Chip and Local_Mems: On-Chip			
High	Low	Moderate		
Ut. Factor	$\simeq 100\%$	$\simeq 100\%$	$\simeq 100\%$	
Control	Simple	Moderate	Complex	
Streaming	Yes	Yes	No	

than the block-based architecture. Finally, the line-based is 5%–55% less energy efficient than the other two architectures.

Summarizing the above results in the following statements.

- In the case that *Img_M* lies off-chip and local memories lie on-chip, the block-based is the most energy-efficient, while the level-by-level is the less energy-efficient architecture.
- When all local memories and *Img_M* are stored on-chip, then the most energy efficient architecture is: 1) either the level-by-level when $((N_W > 11) \wedge (N \times M < 2^{19})) \vee ((5 < N_W \leq 11) \wedge (N \times M < 2^{17}))$ or 2) the block-based when: $(N_W \leq 5) \vee (N \times M \geq 2^{19})$.

D. Discussion

From the comparison performed for the three filters of JPEG 2000, it is evident that the efficient and secure selection of one, among the alternative architectures, requires a detailed exploration of the various tradeoff. None of the alternative architecture has a clear lead in all cases and/or for all sets of parameters. The comparison result turns in favor of the architecture that, for a certain implementation platform and a certain type of filter and decomposition level of the 2-D-DWT, combines low integration cost (related to number of filtering units and memory requirements), sufficient throughput for the given task, and dissipates less energy. This sort of exploration is facilitated by the derived formulas (Table III) and conclusions of this paper, the most important of which are summarized in Table IV.

Specifically, the exploration performed in Section VII-A–C indicated that, in cases that technology and cost allow for the on-chip integration of *Img_M*, the easiest to implement level-by-level architecture combines relatively high throughput and low energy dissipation, while requiring the smallest amount of storage. However, if *Img_M* is stored off-chip (which today is the typical case), then the level-by-level is transformed to the slowest and most energy-hungry among the architectures studied here.

In the latter case and for relatively small filter widths, the line-based architecture offers high throughput at expense of in-

creased energy dissipation, while the block-based architecture requires the lowest energy budget at the expense of processing speed. Furthermore, in the case of relatively large filter widths, the block-based overturns the other two architectures in terms of both energy dissipation and throughput. However, the block-based architecture has the disadvantages of being the hardest to implement, of not being suitable for streaming applications, and of significantly modifying its storage requirements when the number of decomposition levels varies.

VIII. CONCLUSION

In this paper, alternative hardware architectures for the 2-D-DWT have been analyzed and compared in terms of memory requirements, throughput, and energy dissipation. This paper does not cover all architectures proposed in the past, but focuses on these that are likely to be implemented in real-life designs. The comparison is based on theoretically derived formulas for memory requirements, throughput, and energy dissipation. The formulas are generic in terms of parameters of both the 2-D-DWT and the implementation platform. The comparison has indicated that none of the architectures has a clear lead for all sets of parameters, but it has also lead to the identification of strengths and weaknesses of each architecture, and the conditions under which each architecture overturns the others in terms of storage requirements, processing speed, and energy dissipation.

REFERENCES

- [1] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electron. Lett.*, vol. 26, pp. 1184–1185, July 1990.
- [2] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 191–202, June 1993.
- [3] M. Vishwanath, R. M. Owens, M. J. Irwin, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 305–316, May 1995.
- [4] J. T. Kim *et al.*, "Scalable VLSI architectures for lattice structure-based discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 1031–1043, Aug. 1998.
- [5] C. Chakrabarti, M. Vishwanath, and R. M. Owens, "Architectures for wavelet transform: A survey," *J. VLSI Signal Processing*, vol. 4, pp. 171–192, 1996.

- [6] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, "Optimal memory organizations for scalable texture codecs in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 218–242, Mar. 1999.
- [7] S. Mallat, "Multifrequency channel decompositions of images and wavelet models," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 2091–2110, Dec. 1989.
- [8] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*. Norwell, MA: Kluwer, 1998.
- [9] P. Landman, "Low Power Architectural Design Methodologies," Ph.D., Univ. of California, Berkeley, 1994.
- [10] P. E. Landman and J. M. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Transactions VLSI Syst.*, vol. 3, pp. 173–187, June 1995.
- [11] K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in low-power RAM circuit technologies," *Proc. IEEE*, vol. 83, pp. 524–543, Apr. 1995.
- [12] J. M. Rabaey and M. Pedram, *Low Power Design Methodologies*. Norwell, MA: Kluwer, 1995.
- [13] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: Wiley, 1999.
- [14] P. Wen-Shiaw and L. Chen-Yi, "An efficient VLSI architecture for separable 2-D discrete wavelet transform," in *Proc. 1999 Int. Conf. Image Processing (ICIP99)*, vol. 2, Oct. 1999, pp. 754–758.
- [15] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. Image Processing*, vol. 9, pp. 378–389, Mar. 2000.
- [16] *JPEG 2000 Image Coding System V1.0*, ISO/IEC FCD15444-1: 2000, 2000.

Nikos D. Zervas received the Diploma in electrical and computer engineering in 1997 from the University of Patras, Patras, Greece, where he has been working toward the Ph.D. degree in the Department of Electrical and Computer Engineering since 1998.

His research interests are in the area of high-level power optimization techniques and methodologies for multimedia and telecommunication applications.

Mr. Zervas received an award from IEEE Computer Society in the context of the Low-Power Design Contest of 2000 IEEE Computer Elements Mesa Workshop. He is a member of the Technical Chamber of Greece.

Giorgos P. Anagnostopoulos was born in Athens, Greece, in 1973. He received the Diploma in electrical and computer engineering from the University of Patras, Patras, Greece, in 1998. His thesis "Design and Implementation of a NIC for a Wireless LAN transceiver at 2.4 GHz" was awarded by the Technical Chamber of Greece. Since 2000, he has been working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Patras, in the area of low-power VLSI design for multimedia applications.

From 1998 to 2000, he was a Researcher in the area of electronic instrumentation for biomedical applications in the Department of Informatics, Division of Communications and Signal Processing, University of Athens, Athens, Greece. He was also a Developer for the European-founded project Microscanning Endoscope with Diagnostic and Enhanced Resolution Attributes (MEDEA-BIOMED2). He is currently with Pleinlaan2, Department ETRO, Vrije Universiteit Brussel, Brussels, Belgium

Vassilis Spiliotopoulos received the Diploma in electrical and computer engineering in 1998 and the M.S. degree in hardware–software systems integrated design in 1999, both from the University of Patras, Patras, Greece, where he is currently working toward the Ph.D. degree in VLSI design in the Electrical and Computer Engineering School.

His research interests are on memory efficient implementations of signal and image processing algorithms, particularly on image compression.

Yiannis Andreopoulos received the electrical engineering diploma and the M.Sc. degree in signal and image-processing systems from the University of Patras, Patras, Greece. Since October 2000, he has been working toward the Ph.D. degree at Vrije Universiteit, Brussels, Belgium.

His research interests are in the field of image and scalable video-coding, specializing in combined algorithmic and implementation topics for hardware and software systems.

Costas Goutis was a Research Assistant and Research Fellow in the Department of Electrical and Electronic Engineering, University of Strathclyde, Strathclyde, U.K., from 1976 to 1979, and Lecturer in the Department of Electrical and Electronic Engineering, University of Newcastle upon Tyne, U.K., from 1979 to 1985. Since 1985, he has been an Associate Professor and Full Professor in the Department of Electrical and Computer Engineering, University of Patras, Patras, Greece. His recent research interests focus on VLSI circuit design, low-power VLSI design, systems design, analysis and design of systems for signal processing, and telecommunications. He has been awarded a large number of Research Contracts from ESPRIT, RACE, and National Programs.