A Pipeline VLSI Architecture for High-Speed Computation of the 1-D Discrete Wavelet Transform

Chengjun Zhang, Chunyan Wang, *Senior Member*, *IEEE*, and M. Omair Ahmad, *Fellow*, *IEEE*

***Abstract*—In this paper, a scheme for the design of a high-speed pipeline VLSI architecture for the computation of the 1-D discrete wavelet transform (DWT) is proposed. The main focus of the scheme is on reducing the number and period of the clock cycles for the DWT computation with little or no overhead on the hardware resources by maximizing the inter-stage and intra-stage parallelism of the pipeline. The inter-stage parallelism is enhanced by optimally mapping the computational load associated with the various DWT decomposition levels to the stages of the pipeline and by synchronizing their operations. The intra-stage parallelism is enhanced by decomposing the filtering operation equally into two subtasks that can be performed independently in parallel and by optimally organizing the bit-wise operations for performing each subtask so that the delay of the critical data path from a partial product bit to a bit of the output sample for the filtering operation is minimized. It is shown that an architecture designed based on the proposed scheme requires a smaller number of clock cycles compared to that of the architectures employing comparable hardware resources. In fact, the requirement on the hardware resources of the architecture designed by using the proposed scheme also gets improved due to a smaller number of registers that need to be employed. Based on the proposed scheme, a specific example of designing an architecture for the DWT computation is considered. In order to assess the feasibility and the efficiency of the proposed scheme, the architecture thus designed is simulated and implemented on an FPGA board. It is seen that the simulation and implementation results conform to the stated goals of the proposed scheme, thus making the scheme a viable approach for designing a practical and realizable architecture for real-time DWT computation.**

***Index Terms*—Discrete wavelet transform, FPGA implementation, parallel architecture, pipeline architecture, real-time processing, VLSI architecture, multi-resolution filtering, DWT computation, inter- and intra- stage parallelism.**

I. Introduction

SINCE the development of the theory for the computation of the discrete wavelet transform (DWT) by Mallet [1] in 1989, the DWT has been increasingly used in many different areas of science and engineering mainly because of the multi-resolution decomposition property of the transformed signals. The DWT is computationally intensive because of multiple levels of decomposition involved in the computation of the DWT. It is, therefore, a challenging problem to design an efficient VLSI architecture to implement the DWT computation for real-time applications, especially those requiring processing of high-frequency or broadband signals [2]–[4].

Many architectures have been proposed in order to provide high-speed and area-efficient implementations for the DWT computation [5]–[8]. In [9]–[11], the poly-phase matrix of a wavelet filter is decomposed into a sequence of alternating upper and lower triangular matrices and a diagonal matrix to obtain the so-called lifting-based architectures with low hardware complexity. However, such architectures have a long critical path, which results in reducing the processing rate of input samples. On the other hand, the problem of low processing rate is not acute in the architectures that use convolution lowpass and highpass filtering operations to compute the DWT [12]–[19]. These convolution-based architectures can be categorized as single-stage or multi-stage pipeline architectures. The architectures proposed in [12]–[16] are single-stage architectures in which the DWT computation is performed using a recursive pyramid algorithm (RPA) [20] that results in a reduced memory space requirement for the architectures. Lewis and Knowles [12] have designed a simple single-stage VLSI architecture to implement the computation of the DWT without multipliers. Chakrabarti and Vishwanath [13] have proposed a single-stage SIMD architecture that is aimed at reducing the computation time. Grzeszczak *et al*. [14] have proposed a single-stage systolic array architecture with a reduced hardware resource. Cheng and Parhi [15] have proposed a high-speed single-stage architecture based on hardware-efficient parallel FIR filter structures for the DWT computation. The architectures proposed in [17]–[19] are multi-stage architectures in which the tasks of the various decomposition levels of the DWT computation are distributed to a number of pipeline stages. A high-speed multi-stage pipeline architecture, with one stage to carry out the computation of one decomposition level of the DWT, has been proposed by Marino *et al*. [17]. Park [18] has proposed a pipeline architecture using scalable data recorder units (DRU), each requiring a small amount of hardware resources. Masud and McCanny [19] have proposed a method for the design of an efficient, modular and scalable pipeline architecture by using reusable silicon intellectual property (IP) cores for the DWT computation. The pipeline architectures have the advantages of requiring a small memory space and a short computing time, and are suitable for the real-time computations. However, these architectures have some inherent characteristics that have not yet been fully exploited in the schemes for their design. The computational performance of such architectures could be further improved provided that the design of the pipeline makes use the inter-stage and intra-stage parallelism to the maximum

Manuscript received December 15, 2008. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and in part by the Regroupement Stratégique en Microélectronique du Québec (ReSMiQ).

The authors are with the Center for Signal Processing and Communications, Department of Electrical and Computer Engineering, Concordia University, Montréal, QC, H3G 1M8 Canada (e-mail: z_chengj@ece.concordia.ca; chunyan@ece.concordia.ca; omair@ece.concordia.ca).

extent possible, synchronizes the operations of the stages optimally, and utilizes the available hardware resources judiciously.

In this paper, a scheme for the design of a pipeline architecture for a fast computation of the DWT is developed. The goal of fast computation is achieved by minimizing the number and period of the clock cycles. The main idea used for minimizing these two parameters is to optimally distribute the task of the DWT computation among the stages of the pipeline and to maximize the inter- and intra-stage parallelisms of the pipeline.

The paper is organized as follows. In Section II, a matrix formulation for the DWT computation is presented. In Section III, a study is conducted to determine the number of stages required to optimally map the task of the DWT computation onto the stages of the pipeline. Based on this study, in Section IV, a scheme for the design of a pipeline architecture is developed. In Section V, the performance of the pipeline architecture for the DWT computation using the proposed design scheme is assessed and compared with that of other existing architectures. A specific example of designing an architecture for the DWT computation is also considered and the resulting architecture is simulated and implemented on an FPGA board in order to demonstrate the realizability and validity of the proposed scheme. Section VI summarizes the work of this paper by highlighting the salient features of the proposed design scheme and the resulting pipeline architectures.

II. Formulation of the DWT Computation

A. Matrix Formulation

The 1-D DWT of a signal is computed by performing the filtering operation repeatedly, first on the input data and then on the output data, each time after decimating it by a factor of two, for the successive decomposition levels. The filtering operation uses a quadrature mirror filter bank with lowpass and highpass filters to decompose the signal into lowpass and highpass subband signals, respectively. The transform can be expressed using a matrix formulation in order to provide a better insight into the underlining operations of the DWT as well as to facilitate the proposed scheme for the design of the architecture for its computation.

Let the signal be denoted as $\mathbf{S}=[s_1,s_2,...,s_{N-1},s_N]^{\mathrm{T}}$, where N, the number of samples in the input signal, is chosen to be $2J$, J being an integer. Assume that h_i and g_i (i=0,1,…,L−1) are the coefficients of the L-tap lowpass and highpass filters, respectively. Then, by expressing the transform matrices for the lowpass and highpass computations at the jth (j=1,2,…,J) level decomposition as

$$\mathbf{H}^{(j)}=\begin{bmatrix} h_0\ h_1\ h_2\ h_3\cdots h_{L-1} & 0 & 0\ 0\cdots & 0\ 0\ 0\ 0 \\ 0\ 0\ h_0\ h_1\cdots h_{L-3}\ h_{L-2}\ h_{L-1} & 0\cdots & 0\ 0\ 0\ 0 \\ \vdots & \vdots & \\ 0\ 0\ 0\ 0 \quad \cdots & & 0\,h_0\ h_1\ h_2\ h_3 \\ 0\ 0\ 0\ 0 \quad \cdots & & 0\ 0\ 0\ h_0\ h_1 \end{bmatrix} \tag{1a}$$

$$\mathbf{G}^{(j)}=\begin{bmatrix} g_0\ g_1\ g_2\ g_3\cdots g_{L-1} & 0 & 0\ 0\cdots & 0\ 0\ 0\ 0 \\ 0\ 0\ g_0\ g_1\cdots g_{L-3}\ g_{L-2}\ g_{L-1} & 0\cdots & 0\ 0\ 0\ 0 \\ \vdots & \vdots & \\ 0\ 0\ 0\ 0 \quad \cdots & & 0\,g_0\ g_1\ g_2\ g_3 \\ 0\ 0\ 0\ 0 \quad \cdots & & 0\ 0\ 0\ g_0\ g_1 \end{bmatrix} \tag{1b}$$

respectively, where both $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ have a size of $(N/2^j)\times(N/2^{j-1})$, the outputs of the transform at the jth level can be computed from the following:

$$\begin{bmatrix}\mathbf{C}^{(j)}\\ \mathbf{D}^{(j)}\end{bmatrix}=\begin{bmatrix}\mathbf{H}^{(j)}\\ \mathbf{G}^{(j)}\end{bmatrix}\cdot\mathbf{C}^{(j-1)} \tag{2}$$

where $\mathbf{C}^{(j)}$ and $\mathbf{D}^{(j)}$ represent the column vectors of size $N/2^j$ and consist of lowpass and highpass output samples, respectively, at the decomposition level j, with $\mathbf{C}^{(0)}=\mathbf{S}$. It is clear from (1a) and (1b) that the lengths of the filters and the size of the input samples control the number of non-zero entries of the matrices involved, which in turn, determines the complexity of the DWT computation. If the decomposed signals are required to be reassembled into the original form without loss of information, the lowpass and highpass filters must satisfy the perfect reconstruction condition given by

$$g_i=(-1)^{i+1}h_{L-1-i} \tag{3}$$

A border extension of the input signal becomes necessary for the processing of the samples on or near the border of a finite-length signal. There are generally three ways by which the border can be extended in a DWT computation, zero padding, symmetric padding and periodic padding [21]. Even though from the point of view of hardware cost, zero padding is the least expensive, the periodic padding is the most commonly used method for border extension, since it allows a precise recovery of the original signal at or near the border. This method extends the original sequence $\mathbf{S}$ by appending it with its first L−2 samples as

$$\mathbf{S_p}=[s_1,s_2,\cdots,s_{N-1},s_N,s_1,\cdots,s_{L-3},s_{L-2}]^{\mathrm{T}} \tag{4}$$

Thus, in order to operate on the padded input sequence $\mathbf{S_p}$, the transform matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ have to be modified by appending each by additional $L-2$ columns. The elements of the appended columns in a row of a modified transform matrix assume a zero value, if all the filter coefficients already appear in the corresponding row of (1a) or (1b). Otherwise, the elements in the row are made to assume the missing values of the filter coefficients so that all the coefficients appear in that row of the modified transform matrix.

B. Reformulation of (2)

It is seen from (1a) and (1b) that due to the decimation-by-two requirement of the DWT, entries in the successive rows of matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$, and therefore, in their modified versions, are shifted to right by two positions. This property can be utilized to decompose the arithmetic operations in (2) into two parts so that the operations in one part can be performed simultaneously with those of the other one. For this purpose, we now decompose each of the modified transform matrices $\mathbf{H}^{(j)}$ and $\mathbf{G}^{(j)}$ by separating the even and odd numbered columns of each matrix into two sub-matrices. The resulting

sub-matrices, taking into account the perfect reconstruction condition specified by (3), can be expressed as

$$\mathbf{H}_{even}^{(j)} = \begin{bmatrix} h_0\, h_2 \cdots h_{L-2} & 0 & 0 & \cdots & 0 \\ 0\; h_0 \cdots h_{L-4}\; h_{L-2} & 0 & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & \cdots & 0\, h_0\, h_2 \cdots h_{L-2} & 0 \\ 0 & \cdots & 0\; 0\; h_0 \cdots h_{L-4}\; h_{L-2} \end{bmatrix} \tag{5a}$$

$$\mathbf{H}_{odd}^{(j)} = \begin{bmatrix} h_1\, h_3 \cdots h_{L-1} & 0 & 0 & \cdots & 0 \\ 0\; h_0 \cdots h_{L-3}\; h_{L-1} & 0 & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & \cdots & 0\, h_1\, h_3 \cdots h_{L-1} & 0 \\ 0 & \cdots & 0\; 0\; h_1 \cdots h_{L-3}\; h_{L-1} \end{bmatrix} \tag{5b}$$

$$\mathbf{G}_{even}^{(j)} = \begin{bmatrix} h_{L-1}\, h_{L-3} \cdots h_1 & 0 & 0 & \cdots & 0 \\ 0 & h_{L-1} \cdots h_3\, h_1 & 0 & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & \cdots & 0\, h_{L-1}\, h_{L-3} \cdots h_1 & 0 \\ 0 & \cdots & 0 & 0 & h_{L-1} \cdots h_3\, h_1 \end{bmatrix} \tag{5c}$$

$$\mathbf{G}_{odd}^{(j)} = \begin{bmatrix} h_{L-2}\, h_{L-4} \cdots h_0 & 0 & 0 & \cdots & 0 \\ 0 & h_{L-2} \cdots h_2\, h_0 & 0 & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & \cdots & 0\, h_{L-2}\, h_{L-4} \cdots h_0 & 0 \\ 0 & \cdots & 0 & 0 & h_{L-2} \cdots h_2\, h_0 \end{bmatrix} \tag{5d}$$

in which the entries in the successive rows are shifted to right by only one position. With this decomposition of the transform matrices, the DWT computation as given by (2) can be reformulated as

$$\begin{bmatrix} \mathbf{C}^{(j)} \\ \mathbf{D}^{(j)} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{even}^{(j)} \\ \mathbf{G}_{even}^{(j)} \end{bmatrix} \cdot \mathbf{C}_{even}^{(j-1)} + \begin{bmatrix} \mathbf{H}_{odd}^{(j)} \\ \mathbf{G}_{odd}^{(j)} \end{bmatrix} \cdot \mathbf{C}_{odd}^{(j-1)} \tag{6}$$

where $\mathbf{C}_{even}^{(j)}$ and $\mathbf{C}_{odd}^{(j)}$ are the two sub-vectors consisting of even and odd numbered samples, respectively, in the padded vector of $\mathbf{C}^{(j)}$.

It is seen from (6) that the operations in each of the two terms are identical, and also, they can be performed independently in parallel. Furthermore, in view of the structures of the decomposed transform matrices as given by (5a)–(5d), the filtering operation can be carried out by employing the conventional clocking mechanism used for implementing digital systems.

III. Choice of a Pipeline for the 1-D DWT Computation

In a pipeline structure for the DWT computation, multiple stages are used to carry out the computations of the various decomposition levels of the transform. Thus, the computation corresponding to each decomposition level needs to be mapped to a stage or stages of the pipeline. In order to maximize the hardware utilization of a pipeline, the hardware resource of a stage should be proportional to the amount of the computation assigned to the stage. Since the amount of computations in successive decomposition levels of the transform get reduced by a factor of two, two scenarios can be used for the distribution of the computations to the stages of a pipeline. In the first scenario, the decomposition levels are assigned to the stages so as to equalize the computations carried out by each stage, that is, the hardware requirements of all the stages are kept the same. In the second scenario, the computations of the successive decomposition levels are assigned to the successive stages of a pipeline, on a one-level-to-one-stage basis. Thus, in this case, the hardware requirement of the stages gets reduced by a factor of two as they perform the computations corresponding to higher-level decompositions.

Fig. 1 shows a stage-equalized pipeline structure, in which the computations of all the $K=\log_2 N$ levels are distributed equally among the M stages. The process of stage equalization can be accomplished by dividing equally the task of a given level of decomposition into smaller subtasks and assigning each such subtask to a single stage and/or by combining the tasks of more than one consecutive level of decomposition into a single task and assigning it to a single stage. Note that generally a division of the task would be required for low levels of decomposition and a combination of the tasks for high levels of decomposition.

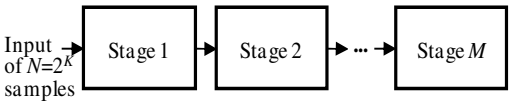

Fig. 1. Stage-equalized pipeline structure.

In a one-to-one mapped structure, the computations of K decomposition levels are distributed exactly among K stages, one level to one stage. In practical applications, a structure with less than K stages is used for the computation of a K-level DWT, as shown in Fig. 2. In this structure, the computations of the first $I-1$ levels are carried out by the stages $i=1, 2, \cdots, I-1$, respectively, and those of the last $K-I+1$ levels are performed recursively by the Ith stage. The amount of hardware resources of a stage is one-half of that of its preceding one except for the Ith stage that has the same size as that of the preceding stage.

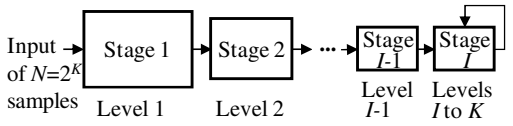

Fig. 2. A one-to-one mapped pipeline structure with I ($I<K$) stages.

The structures of Fig. 1 and Fig. 2 can be used to perform the computations of multiple levels of decomposition. The computation of each level is performed as an L-tap FIR filtering operation by summing the L products of the input samples and the filter coefficients, as described by (2). Generally, one MAC cell is used to carry out one multiplication of an input sample by a coefficient followed by one accumulation operation. In order to perform an uninterrupted L-tap filtering operation with easy control, one can thus use a network of L basic units of such a MAC cell. Since all the decomposition levels perform L-tap filtering operations, it would be desirable that each decomposition level performs its filtering operation using this same type of MAC-cell network. However, in the context of one-to-one mapped pipeline structure of Fig. 2, in which the requirement is that the hardware resource should get reduced by a factor of two from one stage to the next, the use of the same MAC-cell network for all the stages would not be possible unless the pipeline has only two stages. In other words, the first

stage performs the level-1 computation and the second stage performs the computations corresponding to all the remaining levels recursively. In the context of a stage-equalized pipeline structure of Fig. 1, where the requirement is that all the stages should have the same hardware resource, the same MAC-cell network can be used easily for all the stages. However, in this case, the same amount of the computations cannot be assigned to all the stages that are based on the same MAC-cell network unless again there are only two stages in the pipeline.

In a situation of a pipeline of more than two stages, each based on a network of L MAC cells, one cannot achieve a resource-efficient architecture. Thus, for either pipeline structure, i.e., the one-to-one mapped or stage-equalized, a two-stage pipeline would be the best choice in terms of the hardware efficiency as well as from the standpoint of design and implementation simplicity. Note that the two-stage version of either pipeline structure is the same and it is shown in Fig. 3. An additional advantage of the two-stage pipeline is in the design flexibility of a MAC-cell network where the multiplication and accumulation operations can be furnished together by using logic gates. These logic gates could be arranged into more efficient arrays yielding a shorter propagation delay for the MAC-cell network. Based on the above discussion, it seems logical to use the two-stage pipeline structure of Fig. 3 for the design and implementation of an architecture for the 1-D DWT computation. The next section is concerned specifically with a detailed design of the architecture.

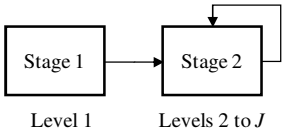

Fig. 3. Pipeline structure with two stages.

IV. Architecture Design

In the previous section, we advocated a two-stage pipeline structure for the computation of the 1-D DWT. The structure, whose development is constrained by the nature of the DWT computation, is capable of optimizing the use of hardware resources. In this two-stage structure, stage 2 performs by operating on the data produced by stage 1 as well as on those produced by itself, and therefore, the operations of the two stages need to be synchronized in a best possible manner [22]. In this section, we present the design of the proposed two-stage pipeline architecture focusing on data synchronization, the details of the various components comprising the stages, and inter and intra stages data flow.

A. Synchronization of Stages

In order to develop a suitable synchronization scheme, consider the timing diagram for the relative operations of the two stages shown in Fig. 4, where t_1 and t_2 are the times taken individually by stage 1 and stage 2, respectively, to carry out their operations, and t_a and t_c are the time spans during which stage 1 or stage 2 alone is operational, and t_b is the overlapped time span for the two stages. Our objective is to minimize $t_a+t_b+t_c$. Since the operation of stage 1 is independent of that of stage 2, it can continue its operation continuously until the computation of all the samples of decomposition level 1 are computed. In Fig. 4, the slots shown for stage 1 correspond to $N/2$ samples of decomposition level 1 that it has to compute. The presence of continuous slots indicates that stage 1 can continue its operation uninterruptedly without having any idle slot. Thus, the minimal possible value for t_1 is equal to $N\cdot T_c/2$, where T_c is the time required to compute one output sample. If $J=\log_2 N$ and we assume that the DWT operation has to be carried out for all the J levels, then the number of samples that stage 2 has to compute is $N/2-1$. Thus, the lowest bound for t_2 is $(N/2-1)T_c$. Now, by choosing a value of t_c equal to its lowest bound, if one can show that $t_2=t_1-T_c$ (i.e. stage 2 does not have any idle slot during t_2), then indeed not only $t_a+t_b+t_c$ will be minimized but one also achieves its lowest bound. Now, we will show that for the proposed architecture this is so possible.

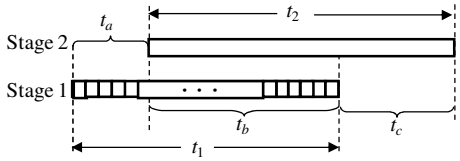

Fig. 4. Timing diagram for the operations of two stages.

Let us first determine the lowest bound on t_c. Since the last sample of level 1 as produced by stage 1 becomes available only at the end of t_b, a sample at level $j\geq 2$ that depends on this last sample directly or indirectly could not possibly be computed during the time span t_b, and therefore, has to be computed during t_c. Assume that (i) during t_c we compute n_c samples of levels 2 and higher, which could not possibly be computed during t_b, and (ii) other output samples necessary for computing those n_c samples have already been computed during t_b. The lowest bound on t_c is n_cT_c. Therefore, in order to compute this bound, we need to determine the value of n_c. The last sample of level 1, which is computed at the end of t_b, is $C_{N/2}^{(1)}$. There are $k=\lceil L/2\rceil$ output samples at level 2 that depend on this sample and they are given as $C_i^{(2)}$, $i=\lfloor(2^{J-1}-L+2)/2\rfloor,\ldots,2^{J-2}$, where $\lceil x\rceil$ and $\lfloor x\rfloor$ represent the smallest integer larger than or equal to x and the largest integer less than or equal to x, respectively. Next, at level 3, there are $\lceil(k+L-2)/2\rceil$ output samples that indirectly depend on $C_{N/2}^{(1)}$ and they are given as $C_i^{(3)}$, $i=\lfloor(2^{J-2}-k-L+4)/2\rfloor,\ldots,2^{J-3}$. Similarly, we can determine the numbers and samples that depend indirectly on $C_{N/2}^{(1)}$ for other levels. Table I givens the listing of the numbers and samples of levels from $j=2$ to J that depend on $C_{N/2}^{(1)}$. After adding the expression in the third column of this table and some manipulation, it can be shown that the value of n_c can be obtained as

$$n_c=\left\lceil\frac{L}{2}\right\rceil+\sum_{j=3}^{J-\lceil\log_2 L\rceil}\left\lceil\left(\left\lceil\frac{L}{2}\right\rceil+\left(2^{j-2}-1\right)L-3\cdot 2^{j-3}+1\right)\Big/2^{j-2}\right\rceil+\left(2^{\lceil\log_2 L\rceil}-1\right) \quad (7)$$

TABLE I
NUMBERS AND SAMPLES IN THE LOWEST BOUND

Level	Samples computed in t_c, $C_i^{(j)}$	Numbers of samples in t_c
2	$i=\lfloor 2^{J-2}-L/2+1\rfloor,\ldots,2^{J-2}$	$k=\lceil L/2\rceil$
3	$i=\lfloor 2^{J-3}-(k+L-4)/2\rfloor,\ldots,2^{J-3}$	$\lceil (K+L-2)/2\rceil$
⋮	⋮	⋮
j	$i=\left\lfloor 2^{J-j}+\frac{L-k-1}{2^{j-2}}+\frac{5-2L}{2}\right\rfloor,\ldots,2^{J-j}$	$\left\lceil \frac{k-L+1}{2^{j-2}}+\frac{2L-3}{2}\right\rceil$
⋮	⋮	⋮
J	$i=2^{J-J}$	1

In Fig. 4, t_a is chosen to be $(n_c+1)T_c$. Next, we explore the possibility of developing a synchronization scheme for computing all the output samples in the context of Fig. 4 with the objective that stage 2 does not create any idle slots. In developing such a scheme, one has to take into consideration, the requirement of the underlying filtering operation of the wavelet computation. This filtering operation imposes the constraint that the first output sample at level j cannot be computed until L samples at level $j-1$ have already been computed and each of the subsequent samples at level j cannot be computed unless two new samples at level $j-1$ have already been computed. Note that this requirement of the filtering operation imposes a constraint on the operation of stage 2 only, since stage 1 operates sequentially and unilaterally to compute the level-1 output samples only. Under this constraint, we now give three steps of the synchronization that govern the computation of the output samples at various decomposition levels by stage 1 and 2.

Step 1. Stage 1 operates continuously to compute the level-1 output samples sequentially.

Step 2. Stage 2 starts the computation of level-2 samples beginning at the time slot (n_c+2).

Step 3. (*a*) *When stage 2 is computing an output sample at the lowest incomplete level $j\geq 2$.*
After completing the computation of the present sample at this level stage 2 moves on to the computation of a sample at the lowest higher level, if the data required for the computation of this sample have become available; otherwise stage 2 continues with the computation of the next sample at the present level j.
(*b*) *When stage 2 is computing an output sample at a level other than the lowest incomplete level.*
After completing the computation of the present sample, stage 2 moves its operation to the lowest incomplete level.

The rationale behind *Step* 3(a) is that moving the operation of stage 2 to a higher level allows more data from level 1 as produced by stage 1 to become available, since the availability of the output samples of level 1 is crucial for the computation of the samples at higher levels. On the other hand, the rationale behind *Step* 3(b) is that there are always more samples to be computed at lower levels than that at higher levels, and therefore, more time needs to be spent in computing lower level samples.

The nature of the filtering operation coupled with the decimation by a factor of 2 requires that, in order for stage 2 to compute a level-2 sample at slot m, stage 2 needs L level-1 samples computed by stage 1 at slots $i+1$, $i+2,\ldots$, $i+L$ ($i<m-L$), of which the samples produced at the last two slots must not have been previously used for the computation of level-2 samples. If stage 2 can meet this requirement during the entire time span t_b, then it can continue its operation uninterruptedly without creating an idle slot. We will now show that, based on the steps presented above, stage 2 would indeed be able to meet this requirement. For this purpose, consider an algorithm, Algorithm 1, which synchronizes the operation of stage 2 during the time span t_b. In this algorithm, we have made use of two counters, namely p and q. The counters p and q represent the total number of samples having been computed at level 2 and that at the levels higher than 2, respectively, at a particular instant of stage-2 operation. Note that at the time that stage 2 starts its operation, stage 1 has already produced n_c+1 level-1 samples. Since a length-L filtering operation would require L input samples and $(n_c+1)>L$, stage 2 not only can start the computation of level-2 samples, but it can continue the computation of the succeeding level-2 samples at least for some time. Since the computation of each level-2 sample makes use of two new level-1 samples during the time in which only one level-1 sample is produced by stage 1, the number of available level-1 samples is reduced by one after the computation of each level-2 sample. However, since stage 2, following *Step* 3 of the synchronization, is allowed to compute the samples at levels higher than 2 without making use of the samples from level 1, the reservoir of level-1 samples is increased by one after the computation of one such a higher-level sample. Therefore, at a particular time, there are $n_b=n_c+1-(p-q)$ level-1 samples available to be used by stage 2 for the computation of the succeeding level-2 samples. Since p increases faster than q, $p-q$ reaches its maximum value at the time slot just before the end of the time span t_b. At this time slot,

$$p=\lfloor 2^{J-2}-L/2\rfloor \tag{8a}$$

$$q=\sum_{i=3}^{J}\left\lfloor \frac{2^{J-2}-\lceil L/2\rceil-(2^{i-2}-1)L+3\cdot 2^{i-3}-1}{2^{i-2}}\right\rfloor \tag{8b}$$

Thus, using (7), (8a) and (8b), the lowest bound of n_b during the time span t_b is calculated as

$$n_b\geq n_c+1-\left(\left\lfloor 2^{J-2}-\frac{L}{2}\right\rfloor-\left(2^{J-2}-1-n_c+\left\lceil\frac{L}{2}\right\rceil\right)\right) \tag{9}$$

Since in practice the filter length L is such that $L<2^{J-1}-1$, the above inequality can be written as

$$n_b\geq\left\lceil\frac{L}{2}\right\rceil-\left\lfloor-\frac{L}{2}\right\rfloor=\begin{cases}L & \text{if } L \text{ is even}\\ L+1 & \text{if } L \text{ is odd}\end{cases} \tag{10}$$

Thus, the lowest bound on n_b is greater than or equal to L. Therefore, during the entire course of the time span t_b, there will always exist sufficient number of samples available to stage 2 for it to continue its level-2 computation in the frame work of Algorithm 1. In other words, during the time span t_b, stage 2 would never have to cease its operation for the lack of availability of at least 2 new level-1 samples, that is, the block

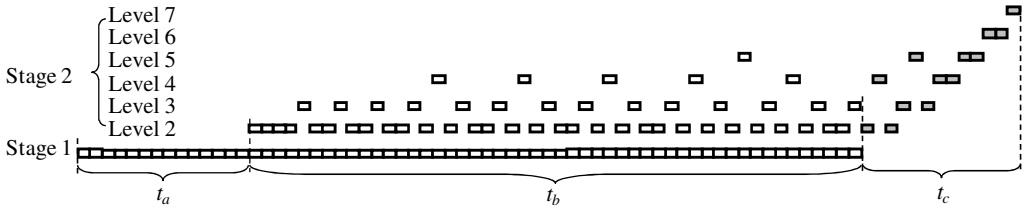

Fig. 5. Synchronization scheme for a 128-point (J=7) DWT computation using length-4 (L=4) FIR filter.

in Algorithm 1 that introduces a unit delay T_c will never be used during the execution of the algorithm.

Algorithm 1: Synchronizing the operation of stage 2 during t_b
Initialize $p \leftarrow 0$, $q \leftarrow 0$
While $p+q \le 2^{J-1}-n_c$
  If (at least 2 new samples available from level 1) **then**
    Compute a new sample at level 2
    $p \leftarrow p+1$
    If (enough data available from the lowest level $k \ge 2$) **then**
    Compute a new sample at level k+1
    $q \leftarrow q+1$
    End if
  Else
    Unit delay T_c
  End if
End while
End algorithm

We now consider an example to illustrate the synchronization scheme that has been presented above. For this purpose, we consider a 128-point (J=7) DWT computation using 4-tap (L=4) FIR filters. The synchronized operation of the two stages is shown in Fig. 5, in which each rectangle represents a time slot during which a lowpass output sample is produced. Stage 1 starts the computation of the first level-1 output sample at slot 1 and continuous its operation until slot 64 when the computation of the 64th level-1 output sample is completed. Equation (7) can be used to obtain the value of n_c as 13. Thus, at the slot number (n_c+2)=15, stage 2 starts the computation of the first level-2 output sample. At this point, the reservoir of level-1 available samples contains (n_c+1)=14 samples. Note that the number of samples in this reservoir decreases by one sample as one new level-2 sample is computed and it increases by one as one sample at a level higher than 2 is computed. However, the general trend is a decline in the number of available level-1 samples from 14 samples at slot 15 to 4 samples at slot 65 when the computations of all level-1 samples are completed. At slot 66, an output sample at level 4 is computed, since the required samples from level-3 have become available for its computation. After this computation, stage 2 returns its operation to the computation of the last level-2 output sample. Note that for the computation of this last level-2 sample, two padded samples would be required, since at this time no level-1 output sample is unused. Beyond this point, all the remaining samples from level 3 to level 7 are computed using *Step* 3 of the synchronization.

B. Design of Stages

Since in the stage-equalized architectures, the two stages together perform the DWT computation with amount and the type of computations of the individual stages being the same, each of the two stages can use identical processing units. However, the control units to be employed by the stages have to be different, since, as seen from Algorithm 1 of the previous subsection, the operation of stage 1 is autonomous, whereas stage 2 must always synchronize its operation with that of stage 1. Based on this algorithm, the design of the control unit used by stage 2 would have to be a bit more involved than that of the control unit used by stage 1. Obviously, in order to synchronize the operation of stage 2 with that of stage 1, a buffer has to be used to store the lowpass output samples from the two stages. Fig. 6 gives a block diagram incorporating all these requirements for the design of the proposed architecture. The two processing units are referred as PU_1 in stage 1 and PU_2 in stage 2. Note that in this architecture, the highpass samples from PU_1 and PU_2 are outputted directly.

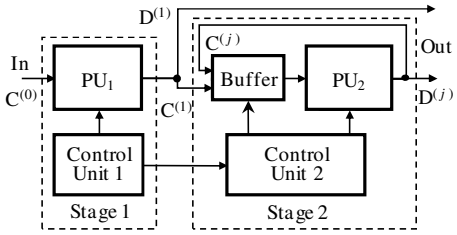

Fig. 6. Block diagram of the two-stage architecture.

In each stage, the processing unit by employing L multiplication-and-accumulation (MAC) cells network performs an L-tap filtering operation and at each clock cycle generates a total of L product terms and their sum. Since, normally, the interval between the two consecutive input samples must not be smaller than the delay of a MAC cell, the maximal allowable data rate of the input to the processing unit would be determined by this delay. However, if the L-MAC-cell network is organized into m sub-networks operating in parallel, the input samples can be applied to these sub-networks in an interleaved manner. The interval of the two consecutive input samples can thus be shortened by a factor m. To this end, considering the problem at hand in which a two-subband filtering operation is performed and for each consecutive decomposition level the input data is decimated by a factor of 2, the L MAC cells can be conveniently organized into a pair of even and odd filter blocks. These even and odd filter blocks, which receive the even and odd numbered input

samples, respectively, employ L/2-MAC-cell networks, and each produces only L/2 product terms and their sums. The partial sums from the two networks are required to be added in an accumulation block by using a carry propagation adder (CPA), as shown in Fig. 7. Since the delay of the accumulate block is comparable to that of the L/2-MAC-cell network, it is useful to pipeline them for parallel computation. Since the high-pass operation differs from that of the low-pass operation only in reversing the sign of the even-numbered coefficients, the proposed organization of the processing unit would allow the filter block to use the same filter coefficients simply by introducing a sign inversion block into the even filter block, as shown in Fig. 7.

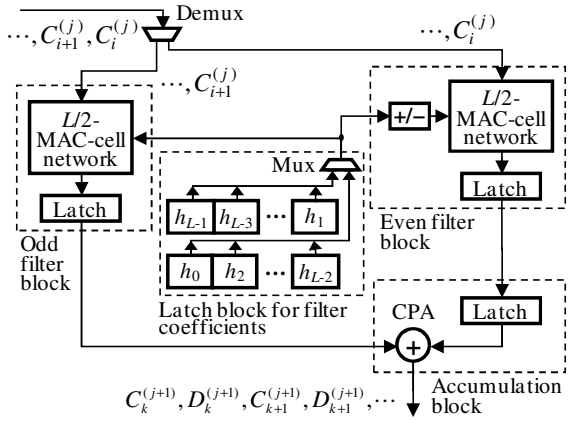

Fig. 7. Block diagram of the processing unit for L-tap filtering computation assuming L to be an even number.

As discussed earlier and seen from Fig. 6, all the output data must be synchronized in accordance with Algorithm 1. This synchronization process is facilitated by introducing in stage 2 a buffer, which stores output data from the two stages and provides input data to stage 2. According to *Step* 2 of the synchronization scheme, during the time span t_a, the number of samples that need to be stored for the operation of stage 2 increases until n_c+1. However, this number will not exceed n_c+1 during the time spans t_b and t_c, since the number of samples newly produced by stage 1 and 2 is equal to or less than that consumed by stage 2. Thus, the minimum capacity of the buffer for the operation of stage 2 is n_c+1 registers. Since the number of output samples at a level that would be needed to compute an output sample at the next higher level will not exceed the filter length L, the buffer, therefore, is divided into $k=\lfloor (n_c+1)/L \rfloor$ channels, as shown in Fig. 8. Each channel consists of L shift registers except channel k that only has (n_c+1 mod L) registers, where (a mod b) is the remainder on division of a by b. Channel 1 is used for storing only the level-1 samples produced by PU_1, whereas channel j=2,...,k for the level-j samples during t_b and t_c, and would also be used for storing the level-1 samples during t_a. Note that channel 2 is also chosen to store the samples of the remaining levels $j \geq k$ since the time slot that all the level-2 samples have been consumed.

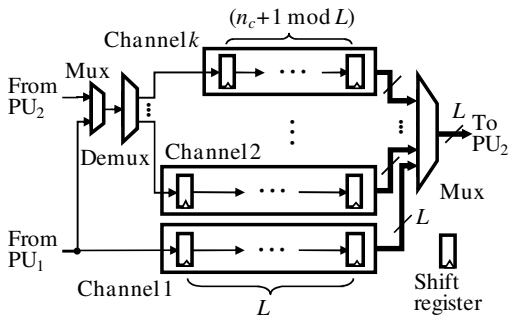

Fig. 8. Structure of the buffer.

C. Design of L/2-MAC-cell Network

In the processing unit shown in Fig. 7, each physical link from a given input bit to an output bit of an L/2-MAC-cell network gives rise to a channel or data path having a delay that depends on the number and the types of operations being carried out along that path [23]. Thus, it is crucial to aim at achieving the shortest critical data path when designing an L/2-MAC-cell network for our architecture. In order to have a better appreciation of the operations of an L/2-MAC-cell network, let us consider an example of the filtering operation of one such network with L/2=2. Let us assume that the input samples and the filter coefficients have the wordlengths of 6 and 3, respectively. Each MAC-cell network has 6 partial products, with a total of 36 bits, which can be produced in parallel, as shown in Fig. 9(a). Our objective is to design a MAC-cell network, in which the bits of the partial products are accumulated in such a way as to optimize the delays of the data paths from the individual bits of the partial products to the output bits of the MAC-cell network.

Even though all the bits of the partial products as given by the array shown in Fig. 9(a) are available simultaneously, they cannot be used in parallel to produce simultaneously all the bits of an output sample. The reason for this is that the processes of accumulation of the bits in each column of the array of the partial products have to be carried out bit-wise and at the same time one has to take care of the propagations of the carry bits. In other words, the accumulation of the partial products has to be carried out in a certain sequence. Thus, the task of accumulation can be divided into a sequence of layers such that the operations of the first layer depend only on the partial products bits and those of the succeeding layers depend on the partial product bits not yet used as well as on the bits of the results of the preceding layers. In order to meet our goal of minimizing the critical path from a partial product bit to a bit of the output sample, we can organize the layers of the MAC-cell network that would carry out the accumulation of the partial products based on the following guiding principle. Minimize the number of layers while minimizing the delay of each layer. The number of layers can be minimized by assigning to each layer the maximum number of such tasks that can be performed independent of each other in parallel. The accumulation task in each layer can be performed by using full-adder (3:2) and double-adder (2×2:3) modules, as shown in Fig. 9(b). The two types of module are chosen, since (i) their delays are about the same so that the delay of any layer can be made to be equal to

this delay irrespective of whether the layer uses one type or two types of modules, and (ii) the two modules can be used together in such a way so that they produces of a smaller number of the propagating carry bits, and therefore, their combined use helps in reducing the number of layers.

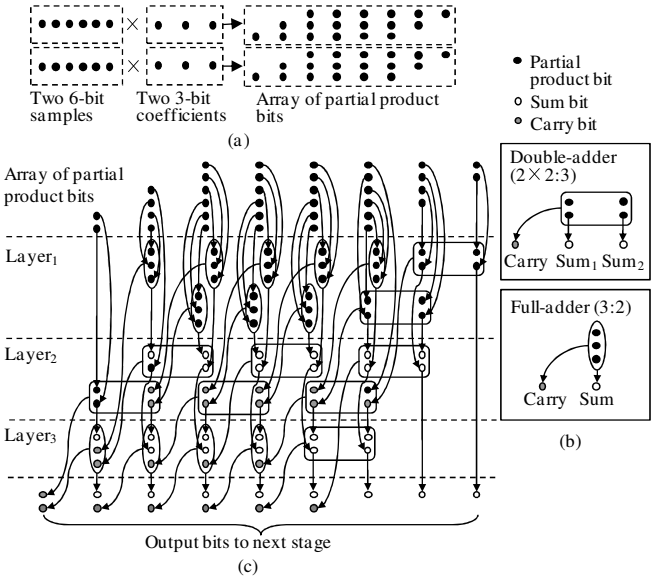

Fig. 9. (a) Formation of an array of partial products. (b) Two types of bit-wise adders. (c) A layered organization of bit-wise addition using the two modules in (b).

With the choice of the combination of the full-adders and double-adders, the first layer can be formed by using as many modules as necessary with the maximum number of partial product bits being utilized as 3-bit or 4-bit inputs to the respective modules. Scanning the partial product array from right to left, a maximum number of bits of this array are first used as inputs to as many full-adder modules as necessary, since in comparison to a double-adder this module is more efficient in consuming the bits of the input array. In this process, whenever in a column (i) only two bits of the partial product array are left unused, these two bits along with a pair of bits from the neighbouring left column of the array are used as inputs to a double-adder modules, and (ii) only one bit of the partial product array is left unused, then this bit is used in the next layer for accumulation. Note that the case of using a double-adder also helps in propagating two carry bits, one internal and the other external to the adder, to the left within the same time delay as that of the full-adder. The next layer can then be formed again by using as many modules as necessary with inputs from the partial product bits, still unused, and the sum and carry output bits from the previous layers being utilized in a carry-save manner. This process can be continued until after the last layer when all the bits of an output sample are produced.

Based on the principles and the procedure enunciated above, we can now give formally an algorithm, Algorithm 2, which carries out the organization of a MAC-cell network, given $L/2$ input samples and $L/2$ filter coefficients. Fig. 9(c) gives an illustration of the organization of the adder modules into three layers of a MAC-cell network for the example considered earlier. It is seen from this figure that the delay of the critical path is equal to that of three full-adders for this particular example.

Algorithm 2: Organizing the bit-wise modules of the MAC-cell network
Initialize an $N_I(k)\times M_I$ array $\mathbf{A_I}$ of partial product bits from the $L/2$ X-bit samples and $L/2$ Y-bit filter coefficients, where $M_I=X+Y-1$ and

$$N_I(k)=\begin{cases} kL/2 & 1\le k\le \min(X,Y)-1 \\ \min(X,Y)\cdot L/2 & \min(X,Y)\le k\le \max(X,Y)-1 \\ (X+Y-k)\cdot L/2 & \max(X,Y)\le k\le M_I \end{cases}$$

While $N_I(k)\ge 3$ for any $1\le k\le M_I$
  Initialize the elements of an $N_O(k)\times(M_I+1)$ array $\mathbf{A_O}$ by $N_O(k)\leftarrow$*zeros* for $k=1,\ldots,M_I+1$
  For every column $i=M_I,\ldots,2,1$
    While $N_I(i)\ge 3$
      Assign 3 bits, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, as inputs to a full-adder
      Append one sum bit to $A_O[{+}{+}N_O(i),i]$, and one carry bit to $A_O[{+}{+}N_O(i-1),i-1]$ in $\mathbf{A_O}$
    End while
    If $N_I(i)=2$ and $N_I(i-1)\ge 2$ **then**
      Assign 2×2 bits, $A_I[N_I(i-1)--,i-1]$, $A_I[N_I(i-1)--,i-1]$, $A_I[N_I(i)--,i]$, $A_I[N_I(i)--,i]$, as inputs to a double-adder
      Append two sum bits to $A_O[{+}{+}N_O(i),i]$, $A_O[{+}{+}N_O(i-1),i-1]$, and one carry bit to $A_O[{+}{+}N_O(i-2),i-2]$ in $\mathbf{A_O}$
    Else
      Carry forward unused bits $A_I[N_I(i)--,i]$ to $A_O[{+}{+}N_O(i),i]$ in $\mathbf{A_O}$
    End if
  End for
  $\mathbf{A_I}\leftarrow\mathbf{A_O}$
End while
End algorithm

Using Algorithm 2, a generalized structure for the MAC-cell network, as shown in Fig. 10, can be generated with $L/2$ X-bit samples and $L/2$ Y-bit filter coefficients as inputs to the network. Layer_0 produces a total of $X\cdot Y\cdot L/2$ partial product bits. The accumulations of these partial product bits are carried out successively by a set of layers of adder modules. A variable size array is used as input to each layer. This array initially contains only the partial product bits, and for successive layers, it contains the sum and carry bits from the previous layers and the partial product bits still unused. An input to a layer that consists of a partial product bit or a sum bit is shown in the figure by an arrow going down vertically into the layer, whereas an input that consists of a carry bit is shown by an arrow going down leftward. The MAC-cell network has a total of $Z=\lceil \log_{3/2}[\min(X,Y)\cdot L/4]\rceil$ layers, which is the minimum number of layers with the choice of using the maximum number of full-adders followed by, if necessary, the double-adders in each layer. The number of adder modules used for each layer progressively decreases from Layer_0 to Layer_Z. The output bits of the MAC-cell network are then used by the accumulation block of the processing unit to produce the final sum. In above design of the MAC-cell network, optimization of its critical path is carried out by incorporating and arranging the multiply and accumulate operations into multiple layers. This leads to a network that has a critical path with a smaller delay than the delay of the MAC cell used in DSP processors, in which the delay of the critical path is simply the sum of the delays associated with a multiplier and an accumulator. The critical

path of the MAC-cell network could be shortened further by encoding the input data to the MAC-cell network using booth encoders. Thus, the delay of the MAC-cell network is reduced by making a smaller number of carry bits to propagate through the MAC-cell network. However, such an improvement can be achieved with an expense of additional hardware resources to be used for encoders.

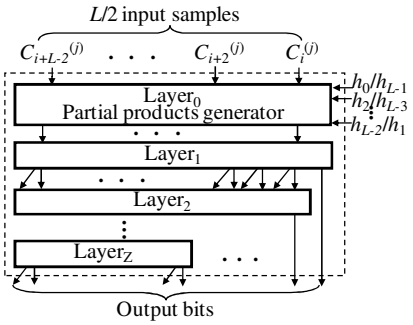

Fig. 10. Structure of the $L/2$-MAC-cell network.

V. Performance Evaluation and FPGA Implementation

In order to evaluate the performance of the architecture resulting from the proposed scheme, we need to make use of certain metrics that characterize the architecture in terms of the hardware resources used and the computation time. The hardware resources used for the filtering operation are measured by the number of multipliers (N_{MUL}) and the number of adders (N_{ADD}), and that used for the memory space and pipeline latches is measured by the number of registers (N_{REG}). The computation time, in general, is technology dependent. However, a metric, that is independent of the technology used but can be utilized to determine the computation time T, is the number of clock cycles (N_{CLK}) consumed from the instant the first sample is inputted to the last sample outputted assuming a given clock cycle period, say unity, as the latency of a MAC cell.

For a J-level DWT computation of an N-sample sequence using L-tap filters, the expressions for the metrics mentioned above for various architectures are summarized in Table II. Assuming that the number of samples N is much larger than $J \cdot L$, it is seen from the table that compared to the architecture of [17], all the other architectures, including the proposed one, require approximately twice the number of clock cycles, except the architecture of [14], which requires four times as many clock cycles. This performance of [17] is achieved by utilizing the hardware resources of adders and multipliers that is four times that required by the architecture of [14] and twice that required by any of the other architectures. However, if the value of $J \cdot L$ cannot be neglected in comparison to that of N, the values of N, J and L should be taken into consideration while comparing the architectures in terms of N_{CLK}. In this regard, only for the proposed architecture and the architecture of [18], N_{CLK} is independent of the filter length with the proposed architecture giving the lowest value of N_{CLK} for a given N. The proposed architecture requires the number of registers that is at least 20% less than that required by any of the other architectures when the filter length L is large. It should be noted that approximately 20% of the hardware resource comprises registers.

Table II
Comparison of Various Architectures

Architecture	N_{MUL}	N_{ADD}	N_{REG}	N_{CLK}
Parallel [13]	$2L$	$2L-2$	$JL+4L$	$N+JL$
Systolic [14]	L	$L-1$	$2JL+L+2$	$2N+2JL$
Pipelined [17]	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$2JL+J+\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-2}} \right\rceil$	$N/2+JL/2$
DRU [18]	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-1}} \right\rceil$	$\sum_{k=1}^{J}\left\lceil \frac{L}{2^{k-1}} \right\rceil - 1$	$JL+2J+\sum_{k=1}^{J}\left\lceil L/2^{k-1} \right\rceil$	$N+2J$
IP core [19]	$J \cdot \lceil L/4 \rceil$	$J \cdot \lceil L/2 \rceil$	$2JL+2J$	$N+JL$
Proposed	$2L$	$2L-2$	$4L+n_c+1$	$N+J$

Since the area of the circuit for the DWT computation depends on the filter length L and the total number of samples N, it would be useful to have a measure of the area of the circuit as functions of L and N. Only the proposed architecture and those of [13] and [18] are used for this study, since the numbers of multipliers and the numbers of adders for these architectures are the same. Thus, any difference in the areas of the three architectures could be accounted for due mainly to the difference in the number of the registers used by each of the architectures. As seen from Table II, the number of registers for the architecture of [13] is $(J+4)L$ and that for the architecture of [18] is approximately $JL+2J+2L=(J+2)L+2J$. However, the number of registers for the proposed architecture not only depends directly on the filter length L but also indirectly on L and N through the parameter n_c. These dependencies are intuitively obvious from the fact that as the filter length or the number of samples increases, the starting point of stage 2 gets more delayed. In other words, n_c is increased. However, it is seen from this figure that the dependence of n_c on N is relatively much more non-linear than its dependence on L. The results of Fig. 11 can be used to obtain a measure of the area of the proposed architecture as functions of L and N. We estimate the areas of the proposed architecture along with that of the other two architectures under the assumption that the ratio of areas of one multiplier, one adder and one register is 12:3:1. The plots of the estimates of the areas as functions of L and N are shown in Fig. 12. It is obvious from this figure that area of the proposed architecture is, in general, lower than those of the other two architectures. The lower area of the proposed architecture can be attributed due mainly to the presence of the parameter n_c in its expression for the N_{REG}. Recall that n_c is a parameter that we minimized in the design of the proposed architecture in order to maximize the parallelism between the two stages, and a lower value of n_c, in turn, results in smaller number of registers required to store the results of the operations of stage 1 before the operation of stage 2 starts.

Considering the clock cycle period T_c as the delay of the MAC cell used by an architecture, the computation time can be obtained as $T=N_{CLK}T_c$. Note that the reciprocal of T_c is simply the throughput of the architecture assuming that one sample is

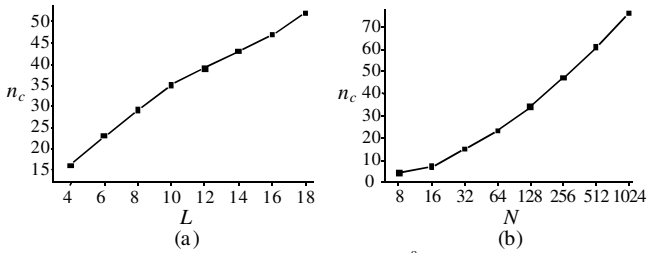

Fig. 11. Estimated values of n_c. (a) n_c vs. L (N=2^8), and (b) n_c vs. N (L=16).

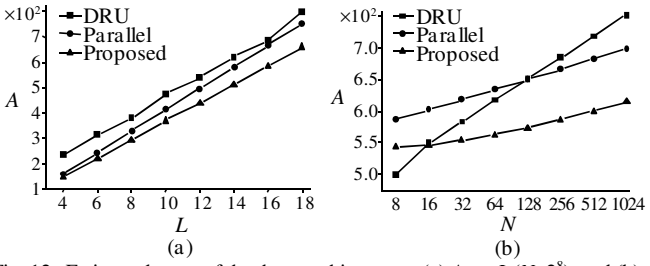

Fig. 12. Estimated areas of the three architectures. (a) A vs. L (N=2^8), and (b) A vs. N (L=16).

inputted during each clock cycle. Using T, one can determine the area-time complexity, AT, where the area, A, mainly comprises the areas of the multipliers, adders and registers. In order to evaluate the performance of the architectures in terms of T_c and AT, we consider an example of designing a circuit for the DWT computation where the sample size N=128 and the number of the decomposition levels J=7. We use Daubechies 6-tap filter (L=6) as analysis filters and the sample and filter coefficient wordlengths are taken as 8 bits. The carry propagation adder of the processing unit utilizes the structure of a combination of carry-skip and carry-select adders [24]. The registers are designed using D-type flip-flops (DFF). All the modules, such as partial products generator, DFF, full-adder, double-adder, multiplexer and demultiplexer, used in the proposed architecture are designed by using 0.35-micron CMOS technology and simulated by using HSpice to obtain the delays. Note that these same modules are also used to evaluate the performance of all the other architectures. Table III shows the values of the clock cycle period and the area-time complexity for the various architectures. It is seen from this table that the proposed architecture has significantly smaller value of the clock cycle period compared to that of all the other architectures. The proposed architecture has the highest throughput of 138 MBPS (megabytes per second) and the lowest area-time complexity, among all the architectures considered.

In order to estimate the power consumption of the proposed architecture, an example of the proposed architecture is constructed for a 7-level DWT computation of 8-bit samples using 6-tap filters and simulated at a clock frequency of 138 MHz using Synopsys Power Compiler. The resulting power consumption values are 154.2 mW and 67.6 mW using 0.35-micron (V_{DD} = 3.3 V) and 0.18-micron (V_{DD} = 1.8 V) technologies, respectively.

In order to have a fair comparison of the power consumption performance of different architectures, the circuit complexities and the technologies used for the circuit design of the architectures under consideration must be the same. In this regard, estimates of the power consumption for the architectures listed in Table III are either unavailable or, if available, the underlying architectures have been designed with substantial differences in the circuit complexities and process technologies. Despite this difficulty in carrying out a fair comparison of power consumption of architectures, we compare the estimated power consumption of the proposed architecture with that given in [25]. The architecture of [25] is also a pipeline architecture that uses the same filter core as that used in [19] of Table III. In [25], an example of the architecture using 9/3 filters and 9-bit samples has been constructed, and simulated for an operation at 100 MHz clock frequency using a 0.35-micron technology. The resulting power consumption figure is 325 mW. This value of power consumption is more than twice the value of 154.2 mW obtained from the example of the proposed architecture in 0.35-micron technology, which is constructed by employing 6-tap filters operating on 8-bit samples at 138 MHz clock frequency.

In order to verify the estimated results for the example of the DWT computation considered above, an implementation of the circuit is carried out in FPGA. Verilog is used for the hardware description and Xilinx ISE 8.2i for the synthesis of the circuit on Virtex-II Pro XC2VP7-7 board. The FPGA chip consists of 36×36 arrays with 11,088 logic cells and it is capable of operating with a clock frequency of up to 400 MHz. The implementation is evaluated with respect to the clock period (throughput) measured as the delay of the critical path of the MAC-cell network, and the resource utilization (area) measured as the numbers of configuration logic block (CLB) slices, DFFs, look-up tables (LUTs) and input/output blocks (IOBs). The resources used by the implementation are listed in Table IV. The circuit is found to perform well with a clock period as short as 8.7 ns, a value that is reasonably close to the estimated value of 7.2 ns. The power consumption of the FPGA chip on which the designed circuit implemented is measured to be 105 mW (V_{DD}=1.5 V). Thus, the simulated value of 67.6 mW is reasonably realistic for power consumption for the circuit realizing the proposed architecture, considering the measured value of the power consumption also includes the power dissipated by the unused slices in FPGA.

TABLE III
EVALUATION OF VARIOUS ARCHITECTURES

Architecture	T_c (ns)	$A \cdot T$
Parallel [13]	17.8	243
Systolic [14]	11.8	141
Pipelined [17]	11.8	183
DRU [18]	10.2	117
IP core [19]	11.8	159
Proposed	7.2	62

TABLE IV
RESOURCES USED IN FPGA DEVICE

Resource	Number used	Total number available	Percentage used
CLB Slices	1532	4928	31%
Flip Flop Slices	858	9856	8%
4-input LUTs	2888	9856	29%
Bonded IOBs	38	248	15%

VI. CONCLUSION

In this paper, a scheme for the design of a pipeline architecture for a real-time computation of the 1-D DWT has been presented. The objective has been to achieve a low computation time by maximizing the operational frequency ($1/T_c$) and minimizing the number of clock cycles (N_{CLK}) required for the DWT computation, which in turn, have been realized by developing a scheme for an enhanced inter-stage and intra-stage parallelisms for the pipeline architecture.

A study has been undertaken that suggests that, in view of the nature of the DWT computation, it is most efficient to map the overall task of the DWT computation to only two pipeline stages, one for performing the task of the level-1 DWT computation and the other for performing that of all the remaining decomposition levels. In view of the fact that the amount and nature of the computation performed by the two stages are the same, their internal designs ought to be the same. There are two main ideas that have been employed for the internal design of each stage in order to enhance the intra-stage parallelism. The first idea is to decompose the filtering operation into two subtasks that operate independently on the even and odd numbered input samples, respectively. This idea stems from the fact that the DWT computation is a two-subband filtering operation, and for each consecutive decomposition level, the input data are decimated by a factor of two. Each subtask of the filtering operation is performed by a MAC-cell network, which is essentially a two-dimensional array of bit-wise adders. The second idea employed for enhancing the intra-stage parallelism is to organize this array in a way so as to minimize the delay of the critical path from a partial product input bit to a bit of an output sample through this array. In this paper, this has been accomplished by minimizing the number of layers of the array while minimizing the delay of each layer.

In order to assess the effectiveness of the proposed scheme, a pipeline architecture has been designed using this scheme and simulated. The simulation results have shown that the architecture designed based on the proposed scheme would require the smallest number of clock cycles (N_{CLK}) to compute N output samples and a reduction of at least 30% in the period of the clock cycle T_c in comparison to those required by the other architectures with a comparable hardware requirement. An FPGA implementation of the designed architecture has been carried out demonstrating the effectiveness of the proposed scheme for designing efficient and realizable architectures for the DWT computation. Finally, it should be pointed out that the principle of maximizing the inter-stage and intra-stage parallelisms presented in this paper for the design of architecture for the 1-D DWT computation is extendable to that for the 2-D DWT computation.

REFERENCES

[1] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 11, no. 7, pp. 674–693, Jul. 1989.

[2] J. Chilo and T. Lindblad, "Hardware implementation of 1D wavelet transform on an FPGA for infrasound signal classification," *IEEE Trans. Nuclear Science*, vol. 55, no. 1, pp. 9–13, Feb. 2008.

[3] S. Cheng, C. Tseng, and M. Cole, "Efficient and effective VLSI architecture for a wavelet-based broadband sonar signal detection system," in *Proc. IEEE 14th Int. Conf. Electronics, Circuits and Systems(ICECS)*, Marrakech, Morocco, Dec. 2007, pp. 593–596.

[4] K.G. Oweiss, A. Mason, Y. Suhail, A.M. Kamboh, and K.E. Thomson, "A scalable wavelet transform VLSI architecture for real-time signal processing in high-density intra-cortical implants ," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 6, pp. 1266–1278, Jun. 2007.

[5] C. Chakrabarti, M. Vishwanath, and R. M. Owens, "Architectures for wavelet transforms: a survey," *J.VLSI Signal Process*, vol. 14, no. 2, pp. 171–192, Feb. 1996.

[6] C. Huang, P. Tseng, and L. Chen, "Analysis and VLSI architecture for 1-D and 2-D discrete wavelet transform," *IEEE Trans. Signal Processing,* vol. 53, no. 4, pp. 1575–1586, Apr. 2005.

[7] M. Martina and G. Masera, "Multiplierless, folded 9/7-5/3 wavelet VLSI architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 9, pp. 770–774, Sep. 2007.

[8] A. Acharyya, K. Maharatna, B.M. Al-Hashimi, and S.R. Gunn, "Memory reduction methodology for distributed-arithmetic-based DWT/IDWT exploiting data symmetry," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 4, pp. 285–289, Apr. 2009.

[9] K.A. Kotteri, S. Barua, A.E. Bell, and J.E. Carletta, "A comparison of hardware implementations of the biorthogonal 9/7 DWT: convolution versus lifting," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 5, pp. 256–260, May 2006.

[10] C. Wang and W.S. Gan, "Efficient VLSI architecture for lifting-based discrete wavelet packet transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 422–426, May 2007.

[11] G. Shi, W. Liu, L. Zhang, and F. Li, "An efficient folded architecture for lifting-based discrete wavelet transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 4, pp. 290–294, Apr. 2009.

[12] A.S. Lewis and G. Knowles, "VLSI architecture for 2D Daubechies wavelet transform without multipliers," *Electron. Lett.*, vol. 27, no. 2, pp.171–173, Jan. 1991.

[13] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: from single chip impelmentations to mapping on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, no. 3, pp. 759–771, Mar. 1995.

[14] A. Grzesczak, M.K. Mandal, and S. Panchanathan, "VLSI implementation of discrete wavelet transform," *IEEE Trans. Very Large Scale Integration Systems*, vol. 4, no. 4, pp. 421–433, Dec. 1996.

[15] C. Cheng and K.K. Parhi, "High-speed VLSI implementation of 2-D discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 56, no. 1, pp. 393–403, Jan. 2008.

[16] S.S. Nayak, "Bit-level systolic imlementation of 1D and 2D discrete wavelet transform," *IEE Proc. Circuits Devices Syst.*, vol. 152, no. 1, pp. 25–32, Feb. 2005.

[17] F. Marino, D. Guevorkian, and J. Astola, "Highly efficient high-speed/low-power architectures for 1-D discrete wavelet transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 47, no. 12, pp. 1492–1502, Dec. 2000.

[18] T. Park, "Efficient VLSI architecture for one-dimensional discrete wavelet transform using a scalable data recorder unit," in *Proc. ITC-CSCC*, Phuket, Thailand, Jul. 2002, pp. 353–356.

[19] S. Masud and J.V. McCanny, "Reusable silicon IP cores for discrete wavelet transform applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 6, pp. 1114–1124, Jun. 2004.

[20] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 42, no. 3, pp. 673–677, Mar. 1994.

[21] M. Ferretti and D. Rizzo, "Handling borders in systolic architectures for the 1-D discrete wavelet transform for perfect reconstruction," *IEEE Trans. Signal Processing*, vol. 48, no. 5, pp. 1365–1378, May 2000.

[22] C. Zhang, C. Wang, and M. O. Ahmad, "An efficient buffer-based architecture for on-line computation of 1-D discrete wavelet transform," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Montreal, Canada, May 2004, vol. 5, pp. 201–204.

[23] C. Zhang, C. Wang, and M.O. Ahmad, "A VLSI architecture for a high-speed computation of the 1D discrete wavelet transform," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kobe, Japan, May 2005, vol. 2, pp. 1461–1464.

[24] A. Satoh, N. Ooba, K. Takano, and E. D'Avignon, "High-speed MARS hardware," in *Proc. 3rd AES conf.*, New York, USA, Apr. 2000, pp. 305–316.

[25] S. Masud and J.V. McCanny, "Rapid design of diorthogonal wavelet transforms," *IEE Proc. Circuits Devices Syst.*, vol. 147, no. 5, pp. 293–296, Oct. 2000.

Chengjun Zhang received the B.S degree and M.S. degree in Physics from Nanjing University, Nanjing, Jiangsu, China, in 1994 and 1997, respectively. He is working toward the Ph.D. degree in the Department of Electrical and Computer Engineering at Concordia University, Montreal, QC, Canada.

His research interests include signal processing, architecture design, and VLSI implementation of digital systems.

Chunyan Wang received the B. Eng. degree in electronics from JiaoTong University, Shanghai, China, and the M. Eng. and Ph.D. degrees from Universite´ Paris Sud, Paris, France.

She joined Concordia University, Montreal, QC, Canada, in 1997, as an Assistant Professor, where she is presently an Associate Professor of Electrical and Computer Engineering.

Her current research areas are low-power analog-mixed VLSI design, CMOS sensor integration, and VLSI implementation of digital signal processing systems.

M. Omair Ahmad (S'69-M'78-SM'83-F'01) received the B.Eng. degree from Sir George Williams University, Montreal, QC, Canada, and the Ph.D. degree from Concordia University, Montreal, QC, Canada, both in electrical engineering.

From 1978 to 1979, he was a member of the Faculty of the New York University College, Buffalo. In September 1979, he joined the Faculty of Concordia University as Assistant Professor of Computer Science. Subsequently, he joined the Department of Electrical and Computer Engineering, Concordia University, where he was the Chair of the department from June 2002 to May 2005 and is presently a Professor. He holds the Concordia University Research Chair (Tier I) in Multimedia Signal Processing. He has published extensively in the area of signal processing and holds four patents. His current research interests include the areas of multidimensional filter design, speech, image and video processing, nonlinear signal processing, communication DSP, artificial neural networks, and VLSI circuits for signal processing. He was a Founding Researcher at Micronet from its inception in 1990 as a Canadian Network of Centers of Excellence until its expiration in 2004. Previously, he was an Examiner of the Order of Engineers of Quebec.

Dr. Ahmad was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART I: FUNDAMENTAL THEORY AND APPLICATIONS from June 1999 to December 2001. He was the Local Arrangements Chairman of the 1984 IEEE International Symposium on Circuits and Systems. During 1988, he was a member of the Admission and Advancement Committee of the IEEE. He has also served as the Program Co-Chair for the 1995 IEEE International Conference on Neural Networks and Signal Processing, the 2003 IEEE International Conference on Neural Networks and Signal Processing, and the 2004 IEEE International Midwest Symposium on Circuits and Systems. He was General Co-Chair for the 2008 IEEE International Conference on Neural Networks and Signal Processing. Presently, he is the Chair of the Montreal Chapter IEEE Circuits and Systems Society. He is recipient of numerous honors and awards, including the Wighton Fellowship from the Sandford Fleming Foundation, an induction to Provost's Circle of Distinction for career achievements, and the award of Excellence in Doctoral Supervision from the Faculty of Engineering and Computer Science of Concordia University.