An Efficient Architecture for 3-D Discrete Wavelet Transform

Triveni.M#1, Manoj Kumar.K*2, Purandara Babu.N#3
P.G. Scholar (M. Tech), Dept. of ECE, AVR & SVR Engineering College, Kurnool, A.P, India
mrudalatriveni@gmail.com afcatmanoj@gmail.com puri.hyd@gmail.com

***Abstract*—This paper presents an architecture of the lifting-based running 3-D discrete wavelet transform (DWT), which is a powerful image and video compression algorithm. The proposed design is one of the first lifting based complete 3-D-DWT architectures without group of pictures restriction. The new computing technique based on analysis of lifting signal flow graph minimizes the storage requirement. This architecture enjoys reduced memory referencing and related low power consumption, low latency, and high throughput compared to those of earlier reported works. The proposed architecture has been successfully implemented on Xilinx Virtex-IV series field-programmable gate array, offering a speed of 321 MHz, making it suitable for real-time compression even with large frame dimensions. Moreover, the architecture is fully scalable beyond the present coherent Daubechies filterbank (9, 7).**

***Index Terms*—Discrete wavelet transform, image compression, lifting, video, VLSI architecture.**

I. Introduction

STILL IMAGE compression technique based on 2-D discrete wavelet transform (DWT) has already gained superiority over traditional JPEG based on discrete cosine transform and is standardized in forms like JPEG2000 [1]. Quite similarly, the application of its 3-D superset, i.e., 3-D-DWT on video, outperforms the current predictive coding standards, like H.261-3, MPEG1-2,4 by rendering the quality features like better peak signal-to-noise ratio (PSNR), absence of blocky artifacts in low bit rates. Furthermore, it has the added provisions of highly scalable compression, which is mostly coveted in modern communications over heterogeneous channels like the Internet [2]. Successful application of 3-D-DWT has been reported in the literature in emerging fields like medical image compression [3], hyper-spectral and space image compression [4], etc. Software-based approaches are experimented to combat the huge computational complexity and memory requirement associated with 3-D-DWT realization [5], [6]. Though the processor speed of modern computers soars high at the order of GHz, data fetching and communicating with external memories consume several T states, making the computation quite slower at the end. As the speeds of the peripherals are still far behind the modern processors, it causes more problems.

Nowadays, most of the applications require real-time DWT engines with large computing potentiality for which a fast and dedicated very-large-scale integration (VLSI) architecture appears to be the best possible solution. While it ensures high resource utilization, that too in cost effective platforms like field-programmable gate array (FPGA), designing such architecture does offer some flexibilities like speeding up the computation by adopting more pipelined structures and parallel processing, possibilities of reduced memory consumptions through better task scheduling or low-power and portability features.

To overcome one of the toughest problems associated with 3-D-DWT architectures—viz., the memory requirement, block based [7], [8] or scan-based architectures [9]–[11] with independent group of pictures (GOP) transform have been reported. However, blocking degrades the PSNR quality while the independent GOPs introduce annoying jerks in video playback due to PSNR drop at transform boundaries [12]. Alternatively, some successful scan-based running transform architectures with convolution filtering have been reported in [13], [14] avoiding these limitations.

After the advent of the lifting scheme [15], [16] in 1994, the computation of DWT has experienced a sea change. While providing facilities like a reduced computational complexity, in-place computation, ease in building nonlinear, and inverse wavelets [16], the lifting also reduces the memory requirement. Thus, it has become a powerful tool to the researchers for computation of both 2-D and 3-D-DWT in several applications. Some lifting-based solely temporal transform techniques with infinite GOPs have been reported in literatures [12], [17], with reduced memory requirements.

Nevertheless, following their attempt to regularize the lifting computation and reduce the storage requirement thereafter, computation of a lifting step is carried out in two stages and performed sequentially. In effect, it doubles the memory referencing and related power consumption while increasing the required processing speed by two fold. Besides, those are merely temporal transform methods; and clearly, there is a gap in the literature for a complete 3-D-DWT architecture which employs lifting and running transform with infinite GOP in its working principle.

Manuscript received March 19, 2007; revised December 4, 2008 and April 8, 2009. First version published September 4, 2009; current version published February 5, 2010. This work was supported by the Ministry of Information Technology, Government of India. This paper was recommended by Associate Editor L.-G. Chen.

A. Das is with the Bangalore Design Center, Nvidia Corporation, Bangalore 560001, India (e-mail: ani.das@gmail.com).

A. Hazra is with STMicroelectronics Private Ltd., Greater Noida, Uttar Pradesh 201308, India (e-mail: anindya.hazra@st.com).

S. Banerjee is with the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India (e-mail: swapna@ece.iitkgp.ernet.in).

Digital Object Identifier 10.1109/TCSVT.2009.2031551

This paper fulfills the requirement herewith presenting a scan-based complete 3-D architecture having infinite GOP. Among the transform components involved in three dimensions, the column and temporal directional transforms are characteristically parallel in nature (for a row-wise scan). The novelty of this paper lies in introducing an ingenious analysis of signal flow graph (SFG), which subsequently shows a newer methodology for computing those parallel transform components with reduced storage overhead. Synchronous data flow and memory arrangements in conjunction with decimated addressing schemes are proposed afterward for incorporating this methodology in hardware. Thus, the designed processor has a minimum memory requirement and much smaller hardware budget with a two-fold throughput and half computing time, latency or memory referencing compared to those of [12], [17]. With a single adder in its critical path, the processor achieves a high speed, which is a fruitful effect of pipelining and incorporation of flipping scheme. Inside the processor, the treatments of the signals at the boundary are done with the mirror extensions proposed in [1].

Section II summarizes the theory of flipping as latest modification on lifting. The proposed architecture along with the analyzed SFG is illustrated in Section III. Section IV discusses the issues related to implementation along with the obtained results after mapping the design in re-configurable Xilinx FPGAs. Besides, a performance comparison with other related works is also furnished in this section. Finally, the paper is concluded in Section V.

II. Theoretical Framework

As the DWT intrinsically constitutes a pair of filtering operations, a unified representation of the polyphase matrix is introduced as follows [16]:

$$P(z) = \prod_{i=1}^{m} \begin{pmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{pmatrix} \tag{1}$$

where $h(z)$ and $g(z)$ stand for the transfer functions for the lowpass and highpass filterbanks, respectively, and all suffixes e and o in the literature correspond to even and odd terms, respectively. Thus, the transform is symbolized with the equation

$$\begin{pmatrix} \lambda(z) & \gamma(z) \end{pmatrix} = \begin{pmatrix} x_e(z) & z^{-1}x_o(z) \end{pmatrix} P(z) \tag{2}$$

with $\lambda(z)$ and $\gamma(z)$ signifying the filtered lowpass and highpass parts of the input $x(z)$.

The lifting scheme [15], [16] factorizes the polyphase representation into a cascade of upper and lower triangular matrices and a scaling matrix which subsequently return a set of linear algebraic equations in the time domain bringing forth the possibility of a pipelined processor. Several other advantages of lifting are mentioned in [16].

For instance, the common Daubechies (9, 7) filterbank can be factorized as

$$P(z) = \begin{pmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{pmatrix} \begin{pmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{pmatrix} \begin{pmatrix} \zeta & 0 \\ 0 & (1/\zeta) \end{pmatrix}. \tag{3}$$

The related algebraic equations are

$$\begin{aligned} s_i^0 &= x_{2i} && \text{(Splitting)} \\ d_i^0 &= x_{2i+1} && \\ d_i^1 &= d_i^0 + \alpha \times (s_i^0 + s_{i+1}^0) && \text{(Predict P1)} \\ s_i^1 &= s_i^0 + \beta \times (d_{i-1}^1 + d_i^1) && \text{(Update U1)} \\ d_i^2 &= d_i^1 + \gamma \times (s_i^1 + s_{i+1}^1) && \text{(Predict P2)} \\ s_i^2 &= s_i^1 + \delta \times (d_{i-1}^2 + d_i^2) && \text{(Update U2)} \\ s_i &= \zeta \times s_i^2 && \text{(Scaling S1)} \\ d_i &= (1/\zeta) \times d_i^2 && \text{(Scaling S2)} \end{aligned} \tag{4}$$

where $\alpha = -1.586134342$, $\beta = -0.05298011854$, $\gamma = 0.8829110762$, $\delta = 0.4435068522$, and $\zeta = 1.149604398$ [16], and also $0 \leq i \leq -1$, L is the data length.

The critical path delay for the above lifting equations is $5T_m + 8T_a$, where T_m and T_a denote the multiplier and adder delay, respectively [18]. The primary reason behind this large delay is stacking of multipliers from the inputs to outputs. To inhibit the effect, the mechanism of flipping has been introduced in [18] which scales the delay down to $3T_m + 4T_a$. As a fruitful result, the processing speed increases significantly when the flipped equations are mapped into hardware.

Following the modification on SFG, the final equations for flipping are

$$\begin{aligned} s_i^0 &= x_{2i} && \text{(Splitting)} \\ d_i^0 &= x_{2i+1} && \\ d_i^1 &= A \times d_i^0 + (s_i^0 + s_{i+1}^0) && \text{(Predict P1)} \\ s_i^1 &= B \times s_i^0 + \frac{(d_{i-1}^1 + d_i^1)}{16} && \text{(Update U1)} \\ d_i^2 &= C \times d_i^1 + \frac{(s_i^1 + s_{i+1}^1)}{2} && \text{(Predict P2)} \\ s_i^2 &= D \times s_i^1 + \frac{(d_{i-1}^2 + d_i^2)}{2} && \text{(Update U2)} \\ s_i &= K0 \times s_i^2 && \text{(Scaling S1)} \\ d_i &= K1 \times d_i^2 && \text{(Scaling S2)} \end{aligned} \tag{5}$$

where $A = (1/\alpha) = -0.630463$, $B = (1/16\alpha\beta) = 0.743750$, $C = (1/32\beta\gamma) = -0.668067$, $D = (1/4\gamma\delta) = 0.638443$, $K0 = (64\alpha\beta\gamma\delta) = 2.590697$ and $K1 = (32\alpha\beta\gamma/\delta) = 1.929981$ (up to six fractional digits) and also $0 \leq i \leq L - 1$, L is the data length [18].

To handle the truncation of the signals at boundaries, mirror extension is utilized by incorporating corresponding changes into (5) at the start and stop of frame sequences and at the individual frame boundaries as well as for the 3-D transforms.

Now, during the computation of 3-D wavelets, the order of spatial and temporal transform components involved can be interchanged where both the arrangements conform to

the definition of 3-D-DWT. However, first temporal and then spatial (t + 2-D) transform suffer from certain limitations with spatial scalability or spatio-temporal decomposition structure [2] which restrict its future extensions. Thus, during the design of the present system, first spatial and then temporal (2-D + t) decomposition are chosen though in due requirement, the reverse method can be equally mapped into hardware without any difficulty.

III. Proposed Architecture

A. Working Principle

Fig. 1 presents the proposed scan-based 1 level 3-D wavelet transform architecture with a block level illustration of principal functional modules. Clearly from the figure, the proposed architecture does the spatial transform first, followed by its temporal counterpart. The following two parts in this section give a detailed view about hand-in-hand working of the different functional blocks to realize those two transform components.

1) *Spatial Transform:* Scanned row-wise with double clock, the incoming frames are fed to the spatial processor (SP) which transform them two dimensionally with the help of two dedicated functional blocks viz., the row processing elements (RPE) and the column processing elements (CPE). As presented in the figure, the scanned pixels are initially fed to RPE for row-transform. On the other hand, CPE remains idle for the starting frame in the video sequence till the initial two rows are transformed by RPE and the processed coefficient blocks are accumulated in line buffers of row MEmory module (RMEM). Having a size of $4N/2$ for a specified frame size of $N \times N$, RMEM provides enough space for the initial two transformed rows. As the transformed coefficients from the third row come out of RPE, column processing commences computation simultaneously by fetching lowpass bands l_0 and l_1 from the memory with the added l_2 band available online. During this phase, the vacant random access memory (RAM) locations of l_0 and l_1 are assigned to l_2 and h_2 coefficient blocks. Thus, after completion of the third row, the fourth one is serially processed, during which the CPE gets busy with the highpass bands of h_0, h_1, and h_2, by fetching all of them from the memory. As the h_0 and h_1 bands are not further utilized in computation, the respective locations are attributed to the storage of l_3 and h_3 bands. The chronology is preserved henceforth, enabling the two processing elements to work in perfect synchronization while spatially transforming each of the frames in sequence. During the computation, CPE requires storage space for some temporary results, which is offered by $6N/2$ depth RAMs of column MEmory module (CMEM). Thus, the SP utilizes an overall memory size of $10N/2$.

With the previously mentioned double scanning, two pixels are fed into SP while two results emerge out of it in every clock cycle, which necessitate a total of $(N^2/2)$ cycles to complete the computation of each frame. If the frames have an even number of rows, both the processing elements run smoothly without any interruption during the skip from one frame to the next. However, for the frame having an odd number of rows, the CPE has to remain idle for $N/2$ cycles (corresponding to one row) at the beginning of each frame to get the first two rows processed. Importantly, in the second instance too, no extra clock cycles are spent by the SP to complete the computation of individual frame.

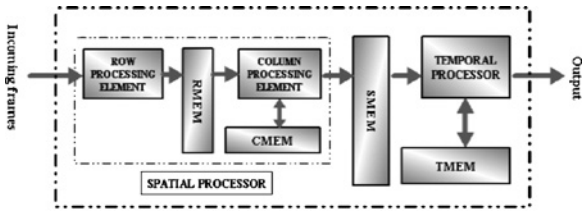

Fig. 1. Proposed architecture RMEM. Line buffers store row processed data CMEM, as well as intermediate results of column processing SMEM. Frame buffers store spatially transformed data TMEM, as well as intermediate results of temporal transform.

2) *Temporal Transform:* The transformed frames, to be decomposed subsequently in the temporal domain, are primarily stored in two dual port frame buffers of spatial MEmory module (SMEM), as shown in Fig. 1. With two such initial frames already stored, and the third one approaching, the temporal processor (TP) starts computing the final transform component of 3-D-DWT. While at every cycle, two pixels of previous two frames are read out from SMEM buffers for the computation, the respective locations are utilized for the reposition of two incoming pixels of the current frame. Nevertheless, the computation necessitates more; one pixel of the third frame is to be read out again from memory at single clock rate, which can be fulfilled through the utilization of the second port of the dual-port RAMs. Thus, the temporal processing proceeds altogether while the RAM locations are refreshed in a continuous manner and after passing a N^2 clock cycles, the current phase of the computation is completed. Additionally, the SMEM buffers are totally filled up with the pixels of the third and fourth frames. In the very next cycle, the corresponding operations of the next phase start in a similar manner with the temporal processor getting busy with the computation of the third, fourth, and fifth frames. The details of the cyclic computation pattern for the temporal and column decompositions are discussed in Section III-C. To incorporate periodicity in the data flow associated with SMEM, efficient memory arrangement and addressing strategy are used. The details are furnished in Section III-E. Like the CPE, the temporal processor produces some temporary results during its operation, which are called back in later cycles repetitively and the temporal memory module (TMEM), which contains three frame buffers inside is used as a storage space for those results. Finally, a group of illustrated pictures is presented in Fig. 2 which helps in understanding the sequence of operations involved in the processing.

Provisions are open for multilevel transform with the current architecture. The simplest possible form will be cascading several 1 level architectures where the *lll* sub-band from one level is passed onto the next. The final output data set then needs to be synchronized or reordered according to the need of a specific encoder.

The following sections discuss the detailed design of principal working modules in the architecture.

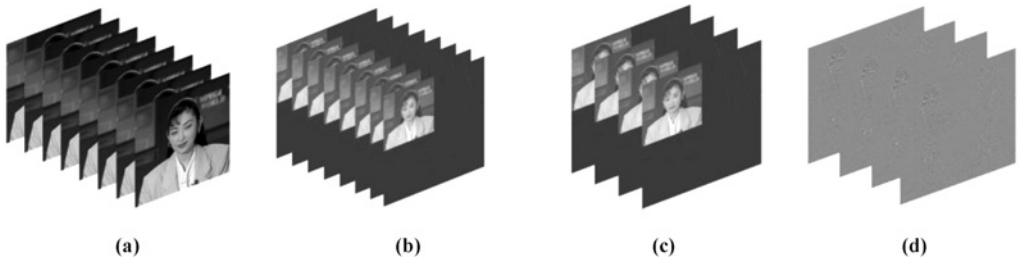

Fig. 2. Spatio-temporal wavelet decomposition with proposed architecture. (a) Original frame sequence. (b) After 1 level spatial transform. (c) Lowpass frames after the temporal transform. (d) Highpass frames after the temporal transform.

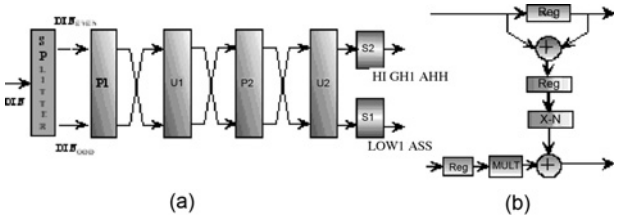

Fig. 3. (a) Architecture of RPE with (b) illustration of a generic P/U module.

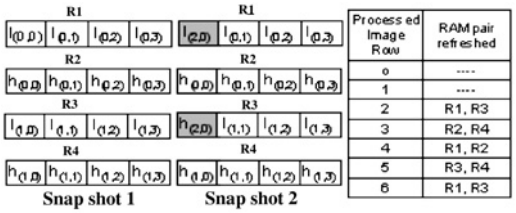

Fig. 4. Two snapshots of RMEM with a model image size of 8 × 8.

B. RPE and the RMEM

Among all the micro-architectures for different submodules, which transform the input video in three directions, the RPE module is the simplest. As described in Fig. 3, it is a straightforward implementation of (5) with pipelining applied to speed up the operations. Scanned with a dual clock, the incoming pixels are separated into successive duos of odd and even ones at the SPLITTER stage and move forward in parallel throughout the pipeline. The required datapath operations of lifting are performed upon these pixels at consecutive Predict (P_i), Update (U_i), and Shift (S_i) stages of the RPE (as depicted in Fig. 3) which finally produces pairs of highpass and lowpass pixels available from the ports OUT EVEN and OUT ODD in a streamlined fashion or manner.

These pixels, prior to column processing, are temporarily put in RMEM which generate the synchronized dataflow to store as well as feed the coefficients to CPE. After processing the initial two rows of a frame the transformed coefficients completely fill up the memory locations as illustrated in snapshot 1 of Fig. 4. At the very next clock cycle, two new pixels viz., $l_{(2,0)}$ and $h_{(2,0)}$, arrive from RPE and they are placed at the locations of R1 and R3 (refer to snapshot 2), which are just left vacant as stored data, namely, $l_{(0,0)}$ and $l_{(1,0)}$ are read out at the commencement of column processing. Subsequent locations are similarly refreshed till all the coefficients from row 2 are stored in those two RAMs. Similarly, during processing of the next row, RAMs R2 and R4 undergo a series of memory refreshments as the locations previously containing h_0 and h_1 coefficient blocks are attributed to the storage of coefficients of h_3 and i_3, available from RPE. Thus, a periodic pattern can be identified among the refreshed RAM pairs, which are further given in a tabular form in Fig. 4 against the processed rows. The proposed memory arrangement is free from any such scenario where the RAM resources would be unnecessarily occupied with stale data which are not to be used for future computation.

C. Analysis of SFG to Facilitate Parallel Computation

The problems associated with designing architectures for column and temporal directional transforms are however critical. In a setup where video frames are scanned row-wise and processed coefficients from RPE are spaced contiguously in rows, the column processor has to wait for an entire row to get another input sample for processing and the temporal processor needs to hold back for the entire frame before it can proceed with the next computation step. Like many other signal processing architectures, the 3-D-DWT processor thus inherently carries a huge memory and latency overhead in its working principle. Clearly, a pipelined design like RPE does not fit in for column and temporal processing and parallel architectures are mostly sought to address this issue. The overall advantage of any DWT processor lies in addressing these performance bottlenecks successfully.

The SFG of lifting, shown in Fig. 5, holds the key for analyzing data dependence inside any DWT processor. Each input and output sample to this SFG denotes a block of data. For row processing, these blocks refer to image pixels adjacently spaced in rows. However, for column transform, each of these blocks signifies a group of processed l and h band coefficients of size $N/2$. Similarly, for temporal transform they relate to 2-D transformed individual frames of N^2 pixels. Thus, when the row processor can freely "sweep across" this graph producing a stream-lined output, an intelligent column or temporal processor must wait and partially finish the computation with available inputs before they proceed to the next step. The key for such parallel processing is to find an optimized basic step of computation which minimizes the latency together with memory overhead for overall architecture.

A careful observation of SFG shown in Fig. 5(a)–(c), infers that the individual slices are the most distinguished representation of the aforesaid basic computation steps. Highlighted in blue, the predict and update calculations inside each such

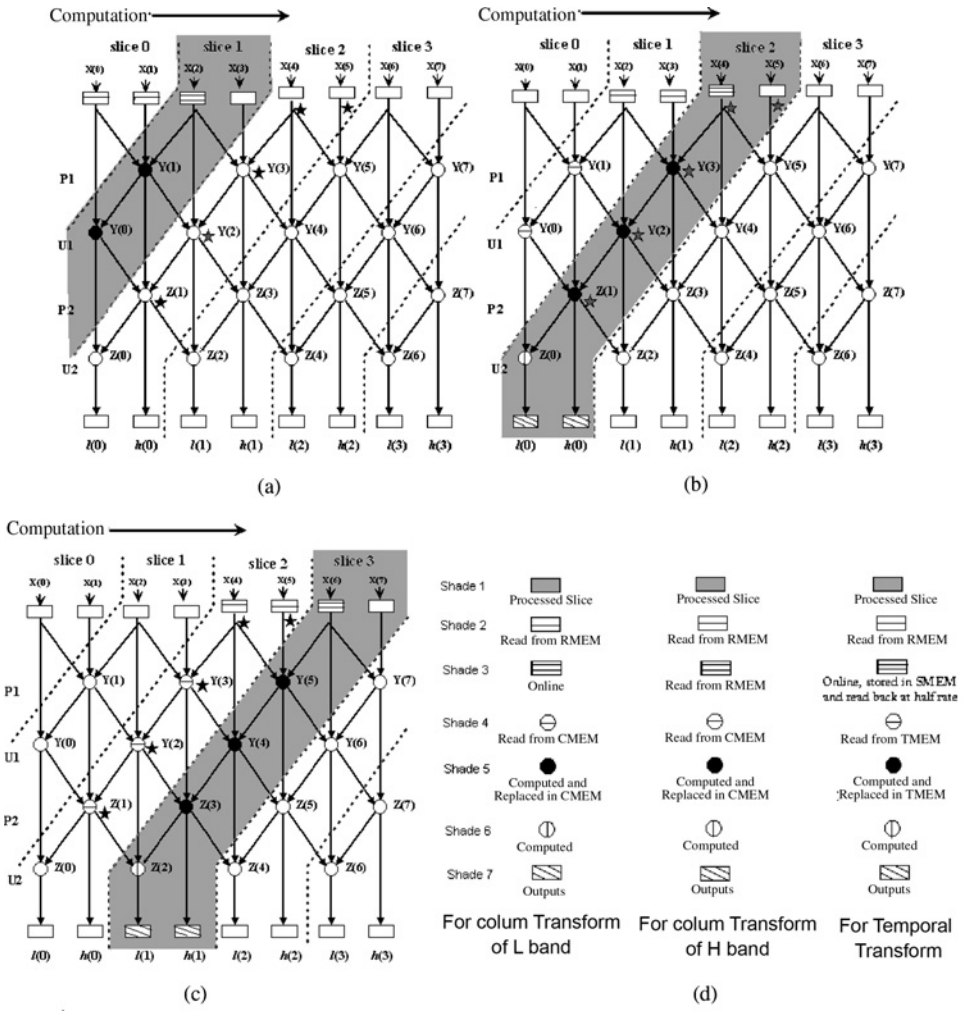

Fig. 5. Data-dependence analysis of SFG for parallel computation. (a) SFG during computation of slice 1. (b) SFG during computation of slice 2. (c) SFG during computation of slice 3. (d) Explanation of color code.

slice can be perfectly represented as the function of two input samples from previous slice (colored in green), one input block from current slice (shown in red) and computed predict and update coefficient blocks from previous slice (highlighted in pink). Since slice 0 holds only input samples, computation should commence with slice 1. Following the sequence in Fig. 5(a)–(c), the computation can henceforth continue for successive slices until the termination of SFG which happens at the end of each frame of column processing or at the termination of video sequence for temporal processing. A pair of output sample blocks (highlighted in brown) from each individual slice are the natural outcomes of this computation.

Such slice-wise computation solves the problem of parallel computing in an efficient way both in terms of latency and memory requirement. From the SFG, the inherent latency to start a basic computation and collect the first output is two and four blocks respectively. This is preserved in proposed slice-wise computation. Counting the contributions from row, column and temporal processors, the latency of the complete 3-D-DWT processor thus adds up to four rows and four frames. The theoretical minimum memory requirement for each parallel computation is five coefficient blocks, which are divided into groups of input coefficients (green) and intermediate results (pink) inside a slice. Thus, the critical memory overhead of entire 3-D-DWT is five rows plus five frames.

The present architecture successfully implements this slice-wise computing strategy carefully preserving the critical latency and memory requirement with the help of some unique memory arrangement techniques. Explained in Fig. 5(d), the individual group of memory blocks is handled differently during the column transform of l and h bands and temporal transform. While for the l band processing, the computation starts with input coefficient block in red reaching on-the-go from RPE and the green ones being fed from RMEM, during the h band processing all three of them are read from RMEM. For the temporal transform, the frame from the current slice is actually stored in SMEM and read back at half rate. The intermediate coefficients in pink are stored in CMEM and TMEM, respectively.

With the description of RMEM as detailed in the previous section, the next ones depict how the column and temporal processors and related memory blocks are designed to preserve the theoretical critical memory requirement and latency in overall architecture.

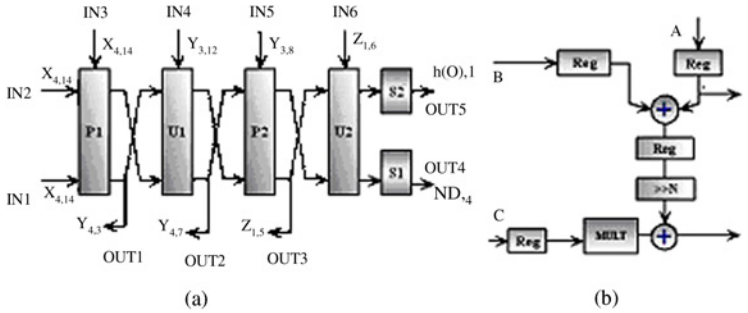

Fig. 6. (a) Snapshot of CPE pipeline and (b) detailed P/U module.

D. CPE

The architecture of CPE, shown in Fig. 6(a), is quite similar to that of RPE. Fig. 6(b) presents its inside details with a P/U module. However, the continuity of RPE pipeline is purposefully broken at several places creating a new set of input and output ports which contribute to the aforesaid parallel computation. Among all the ports, IN 1-3 and OUT 4-5 are connected to RMEM and SMEM respectively and help creating a streamlined dataflow from spatial processor to temporal processor, as depicted in the main architecture (refer to Fig. 1). The other ports, IN 4-6 and OUT 1-3 from RPE are utilized to exchange intermediate coefficients with the six CMEM buffers which is the key to slice-wise computation.

With a view to explaining the computing strategy mentioned in the previous section, a snapshot of the CPE is presented in Fig. 6(a) for lucidity. The snapshot is captured while column processing of slice 3 is ongoing for a random frame of the incoming video sequence. The coefficient indices beside each of the ports in the snapshot indicate the respective locations of those coefficients in the row processed frame matrix. Following those indices, the snapshot can be directly mapped onto slice 3 in SFG. In a similar way, the snapshot can be updated for the next couple of slices and thus it helps to visualize the carrying out of the slice wise computing pattern by the CPE as a whole. To maintain the computation, the aforesaid memory requirement of five nodes (refer to Section III-C) can be mapped here onto five $N/2$ depth coefficient blocks. However, considering the fact that CPE has to process the two l and h bands after the row transform, the requirement is doubled to ten $N/2$ which is retained in the design of RMEM and CMEM.

E. SMEM

Once transformed spatially, the frames are directed to SMEM (refer to Fig. 1), which requires a minimum of two frame buffers for the data management. While the first two frames can be given room in those two frame buffers easily, complexity arises when the third frame arrives from SP and the computation is simultaneously started by the TP. While in every clock cycle, a pixel pair of frame 2 can be allocated into the vacant memory locations from where the two pixels of the frames 0 and 1 have been read out for the computation, the temporal processing methodology demands an extra set of the read operation to be carried out; for collecting the corresponding pixels of frame 2 which act as the third set

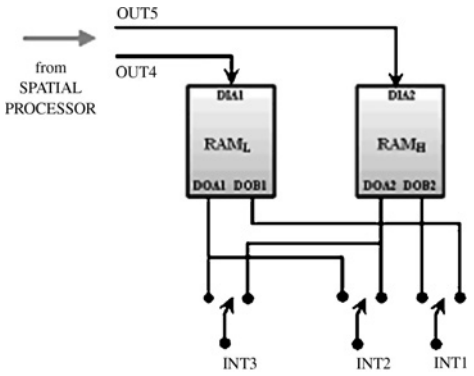

Fig. 7. Arrangement of SMEM frame buffers.

of input in computation of the first lifting step (refer to slice 1 in Fig. 5).

Importantly, this second set of data cannot be provided online to the TP, as the frames are arriving at double rate and computation needs them at single clock. So, all that is needed is to read them back from memory with a half data rate. Thus, the memory arrangement of Fig. 7 is followed where port A of the dual port RAMs is used for reading older frames from memory as well as storing the newer ones in those locations when the Port B remains dedicated for the second set of read operations. Thus, the first kind of operations in effect refreshes the memory with the consecutive duos of frames 0, 1 and 2, 3 and so on, while the second operations are solely responsible for providing the additional pixels of frame $2i$ during computations involving slice i ($i = 1, 2, 3, \ldots$).

As depicted in snapshot 1 of Fig. 8, the frames 0 and 1, being divided in parts L and H, where L and H signify the fact that those pixels emerge from the lowpass and highpass ports of CPE, initially arrange themselves in buffers of SMEM. Once the computation progresses, the pixels of the two frames are replaced with those of frame 2 and as a matter of fact, the new frame gets decimated inside the RAMs as the pixels of the new frame are allocated to memory locations, just released off the older frames every time. Thus, as the computation of slice 2 is completed, the new frames 2 and 3 reposit themselves according to the topography described in snapshot 2 of that figure. The order of decimation increases as the computation moves ahead following a manner very similar to that of fast Fourier transform addressing. The pattern repeats itself after $\log_2\left(N^2\right)$ cycles. Fig. 9 helps us

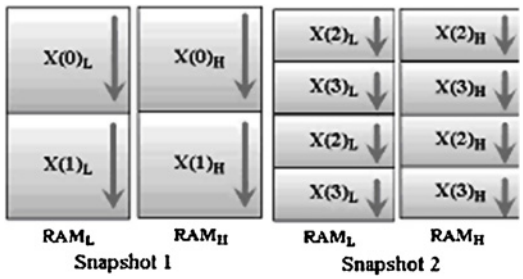

Fig. 8. Two snapshots of the SMEM.

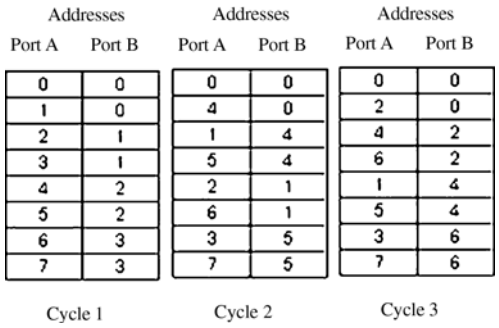

Fig. 9. Addressing pattern in SMEM. RAMs illustrated for a memory depth of 8.

to visualize the addressing pattern for a sample RAM depth of 8.

The dual port BlockRAMs of Xilinx FPGA, which is used as target platform for the proposed architecture, provides the facility of "read before write" operations in the same clock cycle at the memory locations. Utilizing it, simultaneous read and write operations are performed by an address in port A through the channels DIA and DOA, according to the requirements. The address in port B follows the same practice, only changing at a half speed, as they simultaneously pick up two pixels from the RAM pairs in a clock cycle which are further multiplexed to feed INT 2 of TP at single clock rate.

F. TP and TMEM

The architecture of Temporal Processor is identical to that of CPE as both follow the slice-wise computation strategy described in Section III-C. Fig. 10 clarifies the same with a snapshot of pipelined TP unit. The notation X_{fi} $(p \times l = j)$ adopted in the snapshot symbolizes the coefficient X in SFG [refer to Fig. 5(a)–(c)] corresponding to the j^{th} pixel in frame i. Thus, comparison of those frame indices with the SFG reveals the activity of TP in computing the slice 3 nodes at the snapshot instant. The pipelined TP architecture attributes to the difference of pixel numbers across the ports.

While the inputs INT 1-3 are received from SMEM, the TMEM buffers, which act as a storage place for temporary results, exchange these intermediate coefficients with the CPE through ports INT 4-6 and OUT 1-3. The pattern for addressing those frame buffers is straightforward as the serial addressing scheme can handle the data transfer properly.

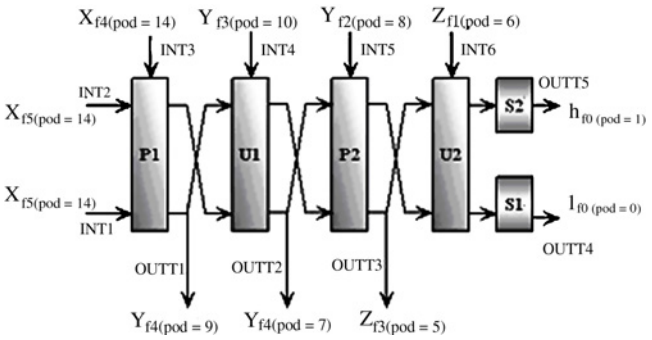

Fig. 10. Snapshot of the TP.

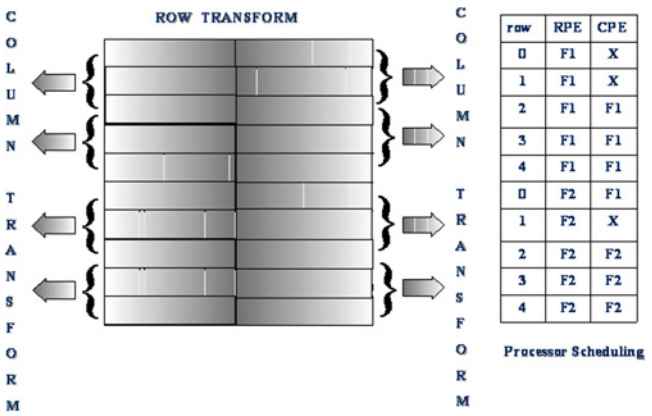

Fig. 11. Processor scheduling during the inter-frame transition.

G. Arrangement to Avoid Latency in Inter-Frame Transition for the SP

At the beginning of computation by SP, which is basically a 2-D-DWT processor, the starting of CPE gets delayed by two rows than RPE. However, large value of cumulative delays for successive frames is to be avoided as it affects the online computation. In the present architecture, this delay is avoided by a processor scheduling arrangement shown in Fig. 11. For the purpose of illustration, a frame depth of 5 has been taken. In the proposed processor scheduling, the CPE has to sit idle at the row 1 of all the frames starting from frame 1 (F1), to collect enough data for column processing. However, the RPE runs smooth, as shown in processor scheduling, indicating no delay between the frames. This is specific to frames containing odd number of rows only. For frames having an even row count, both the CPE and RPE run smoothly over the transition. In that case, the sliced SFG of Fig. 5(c) can be extended over to the next frame, and the CPE computes in slices without interruption. Thus, for each of the cases, inter-frame latency is eliminated.

H. Latency and Complete Memory Requirement

Following the SFG of lifting (refer to Fig. 5), the minimum number of inputs required for computing a wavelet coefficient is restricted to five. Thus, with the provision of the fifth input to arrive online during the computation, the CPE and TP, respectively, need a minimum wait time of four rows ($2N$ clock cycles) and four frames ($2N^2$ cycles) to produce the

first output from the architecture, thereby resulting in a total parametric latency of T clock cycles, where

$$T = 2N^2 + 2N + Latency_{RPE} + Latency_{CPE} + Latency_{TP}. \quad (6)$$

The extra terms arise due to the pipelined design of each computing unit. The memory requirement for five frames and ten $N/2$ length line buffers is $5N^2 + 5N$.

I. Scalability of the Present Architecture and Future Design Trends

Though primarily intended to be an efficient realization of an ideal (2-D + t) DWT in hardware, the present architecture is a scalable one. It can be modified to build hardware engines of some variants of 3-D-DWT and related video compression algorithms reported in recent literatures. The spatial and temporal transform blocks along with their memory modules are completely independent. They can be used either in reverse order with minimal effort or along with some more signal processing blocks to have a fruitful end effect on mapping the architecture to related transform algorithms.

Simply using them in reverse order gives a conventional (t + 2-D) transform. However, we need extra buffers between the temporal processor output and input of the spatial processor. In that case, a complete frame of size $N \times N$ for spatial processing is available from TP in N^2 cycles. To avoid any extra latency and reduce memory requirement, the SP module should be able to complete reading the last pixel of first and successive odd frames as soon as they arrive. Considering the fact that SP is able to read two pixels per cycle, it would add up to an extra latency of $N^2/2$ cycles. During this time, the second and successive even frames arriving parallel (as the TP provides two pixels, one from each processed L and H frames) with their odd counterparts are required to be stored for processing offline in the next $N^2/2$ cycles. Thus, an extra buffer of size of N^2 is required which can be utilized for synchronization between SP and TP. The read-write pattern in this buffer would follow the address generation model of the SMEM buffers, as shown in Fig. 8.

As the present single level architecture adjoins temporal (t) and spatial (2-D) processing blocks, a multilevel transform of $n * (t + 2\text{-}D)$ or $n * (2\text{-}D + t)$ can be achieved by cascading such one-level architectures and passing the output *lll* band from one to the next. Interestingly, the architecture is flexible enough to allow cascading of multiple level stacking of (SP + RMEM + CMEM) and (TP + SMEM + TMEM) inside a single level architecture which realizes (m*t + n*2-D) or (m*2-D + n*t) kind of sub-banding. However, to avoid data loss, frame and row buffers need to be placed between such stacked processor. As mentioned in the above paragraph, the size of such synchronizing RAMs and inter-block latency, and processor synchronizing like Fig. 13 should be calculated by case study.

A combination of motion estimation/compensation, a part of today's popular motion estimated predictive coding standards, with the temporal wavelet transform gives even better energy compaction [23]. Commonly referred to as motion compensated temporal filtering (MCTF), this technique involves incorporating motion compensation in the calculation of lifting steps of (4). A number of approaches [23]–[29] have been devised to realize MCTF in a most beneficial and convenient manner.

The present architecture, focused on improving memory and other performance bottlenecks of a 3-D-DWT, is equally extensible as a MCTF computation engine with all the usual benefits. This necessitates integration of a motion compensation block in the architecture of Fig. 1. The computation of motion vectors inside the new block would require a set of surrounding pixels of temporally connected frames which are already present online inside RAMs of SMEM and TMEM, and can be subsequently fetched. The generated motion vectors need to be forwarded to the TP, which uses them to fetch the motion compensated pixels from SMEM and TMEM during the calculation of Predict (P) and Update (U) stages. A spatial transform would follow subsequently. The overall architecture would be a t + 2-D transform, already mentioned in the above paragraph. The generated motion vectors need to be forwarded along with transformed coefficients to the encoding stage. While computation of motion vectors can grossly benefit from the proposed parallel computing method of temporal wavelet, the new block adds to the memory bandwidth and latency of the present architecture.

IV. Implementation Results and Discussions

A. Multipliers and Datapath Precisions

After the details of the architecture and the data management principles have been thoroughly chalked out, the issues related to mapping the design into a reconfigurable device are of prime interest. These include the precision of the multipliers in the architecture.

Being irrational numbers, the flipping coefficients corresponding to (4) are not ideally realizable in architecture with the hardware multipliers. Instead, those numbers could be considered up to a finite precision during designing. However, the impacts of this limited precision are experienced with lowered PSNR values and subsequent degradation of the quality of reproduced video during the decompression. Additionally, the precision of the data samples right after each multiplier affects the PSNR in a quite similar way.

These facts indicate to a trade-off between the affordable hardware budget, which increases linearly with precision and the video quality. Thus, simulations are done to measure the effects of those two parameters after which respective coefficient and fractional data precision of 11 and 2 bits are fixed up to achieve good video quality at comparatively low hardware constraints. The respective hard multipliers are designed through "shift-n-add" mechanism and pipelined to speed up the processing.

B. Implementation Results

The architecture has been mapped into Xilinx programmable device (FPGA) XC4VFX140 with speed grade of 12 through the Xilinx ISE 7.1i tool. A uniform wordlength of 17 bits has been maintained throughout the processor to afford sufficient data depth. After pipelining the multipliers,

TABLE I
COMPARISON OF THE DESIGNED PROCESSOR WITH EXISTING WORKS

	M. Weeks *et al.*[*1] [9]	Q. Dai *et al.* [10]	C. Parisot *et al.* [19]	B. Das *et al.* [14]	J. Xu *et al.* [12]	Z. Taghavi *et al.* [17]	Proposed
Memory requirement	$2N^2$ $(P+1)+6l$ (spatial + temporal)	$4N^2$ + 896N + 968[*2] (spatial + temporal)	$9N^2$ (only temporal)	$0.5N^2$ + 6N (spatial + temporal)	$5N^2$ (only temporal)	$5N^2$ fast & few slow (only temporal)	$5N^2$ (temporal) + 5N (spatial)
Memory referencing at fixed FPS (i, o/T)[*3]	–	–	–	–	5 ip/T, 5 op/T	5 ip/T, 5 op/T	5 ip/(2T), 6 op/(2T)
Throughput	–	–	–	–	1 res/cycle	1 res/cycle	2 res/cycle
1 Level computational latency	$N^2(P+0.5)+0.5l$	–	–	–	$4N^2$cycles	$4N^2$cycles	$(2N^2)$ cycles
1 level computing time for P frames	$\frac{3}{4}N^2P + \frac{N^2}{2} + \frac{3l}{2}$	–	–	–	$(P+4)N^2$	$(P+4)N^2$	$2N^2 + \frac{P}{2}N^2$
Operating frequency	200 MHz (ASIC)	–	–	100 MHz (0.25 μm BiCMOS)	–	–	321 MHz (Xilinx FPGA – xc4vfx140)
Adders	$6l$ MACs	24 × 9	–	–	–	–	26 × 3
Multipliers	–	24 × 8	–	Nil	–	–	Nil
Area	–	–	–	–	–	–	1825 slices
Hardware utilization	–	–	–	100%	–	–	100%
Filterbank	l-length	For D-9/7	–	D-4	D-9/7	For D-9/7	D-9/7
GOP (P)	$P = 32$ (max)	32 (max)	Infinite	Infinite	Infinite	Infinite	Infinite
Design type	Complete 3-D	Complete 3-D	Temporal processor	Complete 3-D	Temporal processor	Temporal processor	Complete 3-D

[*1] The 3D-DWT-I architecture.

[*2] $4N^2$ corresponds to off-chip memory.

[*3] T symbolizes time period of computation in a standard frame rate.

the critical path for the processor consists of single adder, making it quite fast. A fast counter based controller was designed which handles all the address generation and other switching operations at the high speed of main data-path. Such controllers are programmable and can synchronize the control signal generation according to different video frame sizes. So other than standard $N \times N$, they can handle standard quarter common intermediate format or common intermediate format or various different aspect ratios.

The adders from the library and device dual port block RAMs have been utilized as the principal resources for the designed processor. Simulation is performed by ModelSim XE III 6.0a, which yields a set of end results completely matching the results from MATLAB 7.0.0, where a model of the hardware is created.

The overall design report can be formulated as

Custom frame size	256×256
Group of frames (GOP)	Infinite
Maximum clock frequency	321 MHz
Throughput	Two results/cycle
Initial latency	$2N_2 + 2N\psi + 47$ clock cycles
Number of occupied slices	1776 (2%)
Total number four input LUTs	2188 (1%)
Number of block RAMs	350 (63%).

C. Performance Evaluation

Among the few dedicated architectures for 3-D-DWT reported in literature until date, the inherent problems associated with the designs following block based approaches [7], [8], or having finite GOPs [9]–[11] are already discussed. Amidst the running transform methods, [14] was quite successful being convolution based, whereas [12], [17], though promisingly formulated upon lifting, limit their discussions in the temporal processing methodology alone. Only the present design is one of the first lifting based complete 3-D-DWT architectures which enjoy no restriction on GOP.

A comparison of the present design with those available in literatures is furnished in Table I. It is worth mentioning that though [9], [10], having finite GOPs, do not represent ideal temporal transform.

From the table, the memory requirement of the presented design is found to be less than those of [9], [19] and compared to [10], requirement is less for frame sizes down to the order of 800 pixels. Even so, architecture in [10] necessitates $4N^2$ off-chip memories with higher read-write latency. Memory consumption is low at [14]. However, it implements a lower length D-4 filter bank.

The designs referred in [12], [17] have the same temporal buffer requirement as the proposed one. Nevertheless, in both the cases, dynamic updating of buffers data is done with the arrival of each frame whereas, as mentioned in Sections III-C

TABLE II
COMPARISON OF THE SPATIAL SUBMODULE WITH THREE 2-D DWT ARCHITECTURES

	S. Barua *et al*. [20]	G. Kuzmanov *et al.* [21]	I. S. Uzun *et al.* [22]	Proposed
ASIC/FPGA	Altera FPGA APEX20KE	Xilinx FPGA xc2v1000	Xilinx FPGA Virtex-2000E	Xilinx FPGA xc4vfx140
Speed	66.8 MHz	50 MHz	105 MHz	321 MHz
Area	7726 logic elements	985 slices	3974 slices	973 slices
Memory requirement	$14 \times N/2$: internal memory	22 BlockRAM: total	14 BlockRAM (1 Level DWT)	10 $N/2$ (10 BlockRAM)
Memory access	$\frac{38}{3}N^2(1-\frac{1}{4^L})$	–	–	$\frac{38}{3}N^2(1-\frac{1}{4^L})$
Initial latency	$4N + 35$	–	–	$4N + 28$
Filter used	9/7	4/4	9/7	9/7

and III-E, the proposed methodology succeeds in doing the transform by doing the computations once after the arrival of every two frames only. The subsequent improvements are a halved latency and computational time during processing and a doubled throughput.

Moreover, while working with the same frame rate, the speed burden of the proposed processor and memory referencing become one-half compared to those of [12] and [17]. These together, for such computation intensive processing such as 3-D-DWT, impact the power consumptions to a great extent. Thus, the results indicate the wide possibilities of the proposed design in applications requiring low power consumptions like medical imaging. Also, the computing time is close to 75% of that required by [9] for large GOPs too.

Largely due to the fact that the critical path delay corresponds to a single adder, the maximum operating speed of the architecture reaches 321 MHz which makes it fastest among all. At standard rates of 30 FPS with a frame size of 256×256, any DWT processor with a throughput of 2 requires a minimum 1.09 MHz for real-time 1 level processing. Thus, at 321 MHz, the current design offers quite large computing potentials.

As the spatial processor module is basically a 2-D-DWT engine, comparison of this part is carried out with three other standard 2-D-DWT architectures available in the literatures and presented in Table II. It shows that the present design has the lowest memory requirement among all and latency is lower than that of [20]. Moreover, the proposed architecture is much faster than the rest. The device resource consumption of the present design is also lower than [21] and [22]. Comparing all the performance parameters, this architecture surely gets a cutting edge performance.

V. CONCLUSION

The applications of 3-D wavelet based coding are opening new vistas in video and other multidimensional signal compression and processing. The prominent needs in these diversified application areas are efficient 3-D-DWT engines with good computing power which draws the attention of the dedicated VLSI architectures as the best possible solution. Though the researches of 2-D-DWT architectures are progressing quite fast, fewer approaches are reported in the literatures designing their 3-D counterpart.

This paper has presented a lifting based 3-D-DWT architecture with running transform, possibly the first of its kind. The main flavors of the design are minimized storage requirement and memory referencing, low latency and power consumption and increased throughput, which become evident when they are compared with those of existing ones. Having single adder in its critical path, the mapped processor achieves a high speed of 321 MHz, offering large computing potentials which opens up new vista for real-time video processing applications.

Compared to the original 3-D-DWT transform, successful application of motion compensations before temporal transform has been reported in the literature [2] as a good alternative for predictive coding. It is worth mentioning that the present design is fully scalable to those future modifications and can be accepted as an introductory step toward those future 3-D wavelet computing machines.

REFERENCES

[1] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Process. Mag.*, vol. 18, no. 5, pp. 36–58, Sep. 2001.

[2] J.-R. Ohm, M. van der Schaar, and J. W. Woods, "Interframe wavelet coding: Motion picture representation for universal scalability," *J. Signal Process. Image Commun.*, vol. 19, no. 9, pp. 877–908, Oct. 2004.

[3] G. Menegaz and J.-P. Thiran, "Lossy to lossless object-based coding of 3-D MRI data," *IEEE Trans. Image Process.*, vol. 11, no. 9, pp. 1053–1061, Sep. 2002.

[4] J. E. Fowler and J. T. Rucker, "3-D wavelet-based compression of hyperspectral imagery," in *Hyperspectral Data Exploitation: Theory and Applications*, C.-I. Chang, Ed. Hoboken, NJ: Wiley, 2007, ch. 14, pp. 379–407.

[5] L. R. C. Suzuki, J. R. Reid, T. J. Burns, G. B. Lamont, and S. K. Rogers, "Parallel computation of 3-D wavelets," in *Proc. Scalable High-Performance Computing Conf.*, May 1994, pp. 454–461.

[6] E. Moyano, P. Gonzalez, L. Orozco-Barbosa, F. J. Quiles, P. J. Garcia, and A. Garrido, "3-D wavelet compression by message passing on a Myrinet cluster," in *Proc. Can. Conf. Electr. Comput. Eng.*, vol. 2. 2001, pp. 1005–1010.

[7] W. Badawy, G. Zhang, M. Talley, M. Weeks, and M. Bayoumi, "Low power architecture of running 3-D wavelet transform for medical imaging application," in *Proc. IEEE Workshop Signal Process. Syst.*, Taiwan, 1999, pp. 65–74.

[8] G. Bernabé, J. González, J. M. García, and J. Duato, "Memory conscious 3-D wavelet transform," in *Proc. 28th Euromicro Conf. Multimedia Telecommun.*, Dortmund, Germany, Sep. 2002, pp. 108–113.

[9] M. Weeks and M. A. Bayoumi, "Three-dimensional discrete wavelet transform architectures," *IEEE Trans. Signal Process.*, vol. 50, no. 8, pp. 2050–2063, Aug. 2002.

[10] Q. Dai, X. Chen, and C. Lin, "Novel VLSI architecture for multidimensional discrete wavelet transform," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 8, pp. 1105–1110, Aug. 2004.

[11] W. Badawy, M. Weeks, G. Zhang, M. Talley, and M. A. Bayoumi, "MRI data compression using a 3-D discrete wavelet transform," *IEEE Eng. Med. Biol. Mag.*, vol. 21, no. 4, pp. 95–103, Jul.–Aug. 2002.
[12] J. Xu, Z. Xiong, S. Li, and Y.-Q. Zhang, "Memory-constrained 3-D wavelet transform for video coding without boundary effects," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 9, pp. 812–818, Sep. 2002.
[13] B. Das and S. Banerjee, "Low power architecture of running 3-D wavelet transform for medical imaging application," in *Proc. Eng. Med. Biol. Soc./Biomed. Eng. Soc. Conf.*, vol. 2. 2002, pp. 1062–1063.
[14] B. Das and S. Banerjee, "Data-folded architecture for running 3-D DWT using 4-tap Daubechies filters," *IEE Proc. Circuits Devices Syst.*, vol. 152, no. 1, pp. 17–24, Feb. 2005.
[15] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.*, vol. 3, no. 15, pp. 186–200, 1996.
[16] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 247–269, 1998.
[17] Z. Taghavi and S. Kasaei, "A memory efficient algorithm for multi-dimensional wavelet transform based on lifting," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, vol. 6. 2003, pp. 401–404.
[18] C.-T. Huang, P.-C. Tsneg, and L.-G. Chen, "Flipping structure: An efficient VLSI architecture for lifting-based discrete wavelet transform," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1080–1090, Apr. 2004.
[19] C. Pilrisot, M. Antonini, and M. Barlaud, "3-D scan based wavelet transform and quality control for video coding," *Eur. Assoc. Signal Process. J. Appl. Signal Process.*, vol. 2003, pp. 56–65, Jan. 2003.
[20] S. Barua, J. E. Carletta, K. A. Kotteri, and A. E. Bell, "An efficient architecture for lifting-based two-dimensional discrete wavelet transforms," *VLSI J. Integration*, vol. 38, no. 3, pp. 341–352, Jan. 2005.
[21] G. Kuzmanov, B. Zafarifar, P. Shrestha, and S. Vassiliadis, "Reconfigurable DWT unit based on lifting," in *Proc. Program Res. Integr. Syst. Circuits*, Veldhoven, The Netherlands, Nov. 2002, pp. 325–333.
[22] I. S. Uzun and A. Amira, "Design and FPGA implementation of nonseparable 2-D biorthogonal wavelet transforms for image/video coding," in *Proc. Int. Conf. Image Process. (ICIP)*, vol. 4. Belfast, U.K., Oct. 2004, pp. 2825–2828.
[23] B. Girod and S. Han, "Optimum update for motion-compensated lifting," *IEEE Signal Process. Lett.*, vol. 12, no. 2, pp. 150–153, Feb. 2005.