# A Streamlined Architecture For 3-D Discrete Wavelet Transformation And Inverse Discrete Wavelet Transform

**V Ashok**
M.tech(PG student)
GITAS

**B.Chinna rao**
Prof.&Head,Dept.of ECE
GITAS

**P.M.Francis**
Asst.Prof. in Dept. of ECE
GITAS

## 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-DDWT 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. Further, the digital data can be retrieved using Inverse Discrete Wavelet Transform (IDWT). The images need to be retrieved without loosing of information. The proposed architecture has been successfully implemented on Xilinx Spartan series field-programmable gate array, offering a speed of 40 MHz, making it suitable for realtime 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, IDWT, 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.

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]:

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

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

$$(\ \lambda(z) \ \ \gamma(z)\ ) = (\ xe(z) \ \ z^{-1}xo(z)\ )\ P(z)$$

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

$$P(z) = \begin{pmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{pmatrix} \begin{pmatrix} \zeta & 0 \\ 0 & (1/\zeta) \end{pmatrix}$$
(3)

The related algebraic equations are

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

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

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

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

where $A = (1/\alpha) = -0.630463$, $B = (1/16\alpha\beta) = 743750$, $C = (1/32\beta\gamma) = -0.668067$, $D = (1/4\gamma\delta) = 0.638443$, $K0 = (64\alpha\beta\gamma\delta) = 2.590697$ and $K1 = (32\alpha\beta\gamma/\delta) = 1.929981$ (up to six fractional digits) and also $0 \le i \le L - 1$, $L$ is the data length [18].

To handle the truncation of the signals at oundaries, 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.

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

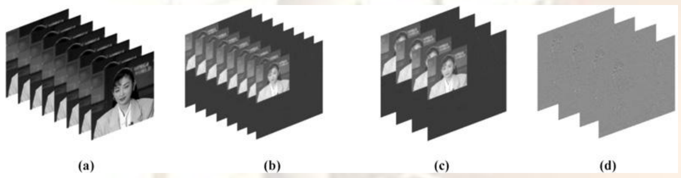


Fig. 2. Spatio-temporal wavelet decomposition with proposed architecture. (a) Original frame sequence. (b) After 1 level spatial transform. (c) Lowpass frames after the temporal transform. (d) Highpass frames after the temporal transform.

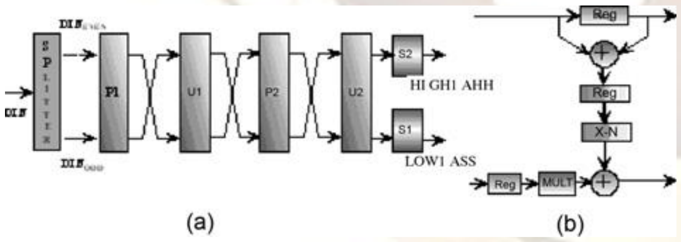
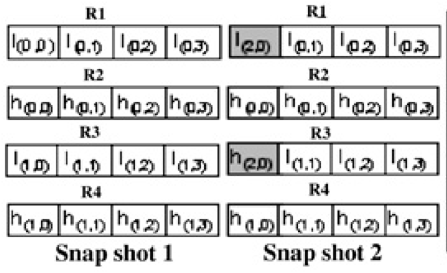


Fig. 3. (a) Architecture of RPE with (b) illustration of a generic P/U module


R1
l(0,0) l(0,1) l(0,2) l(0,3)
R2
h(0,0) h(0,1) h(0,2) h(0,3)
R3
l(1,0) l(1,1) l(1,2) l(1,3)
R4
h(1,0) h(1,1) h(1,2) h(1,3)
Snap shot 1
R1
l(2,0) l(0,1) l(0,2) l(0,3)
R2
h(0,0) h(0,1) h(0,2) h(0,3)
R3
h(2,0) l(1,1) l(1,2) l(1,3)
R4
h(1,0) h(1,1) h(1,2) h(1,3)
Snap shot 2

| Processed Image Row | RAM pair refreshed |
|---|---|
| 0 | ---- |
| 1 | ---- |
| 2 | R1, R3 |
| 3 | R2, R4 |
| 4 | R1, R2 |
| 5 | R3, R4 |
| 6 | R1, R3 |

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 (Pi), Update (Ui), and Shift (Si) 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 $h0$ and $h1$ coefficient blocks are attributed to the storage of coefficients of $h3$ and $i3$, 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

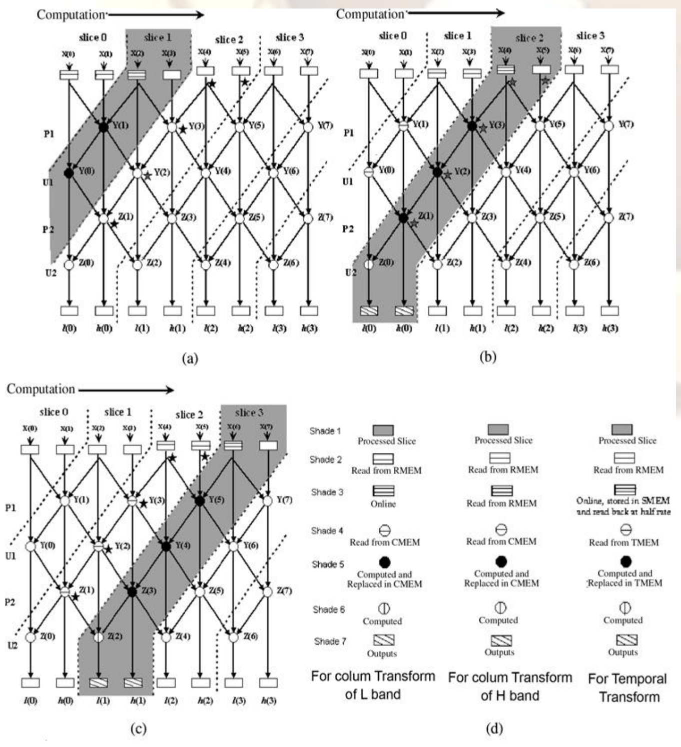


Fig. 5. Data-dependence analysis of SFG for parallel computation. (a) SFG during computation of slice 1. (b) SFG during computation of slice 2. (c) SFG during computation of slice 3. (d) Explanation of color code.

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.

The present architecture successfully implements this slicewise 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.

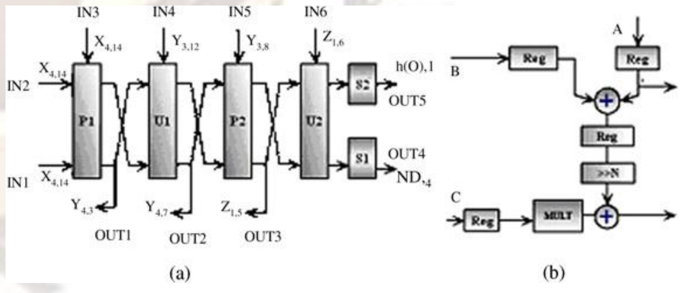


Fig. 6. (a) Snapshot of CPE pipeline and (b) detailed P/U module.

## 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.

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

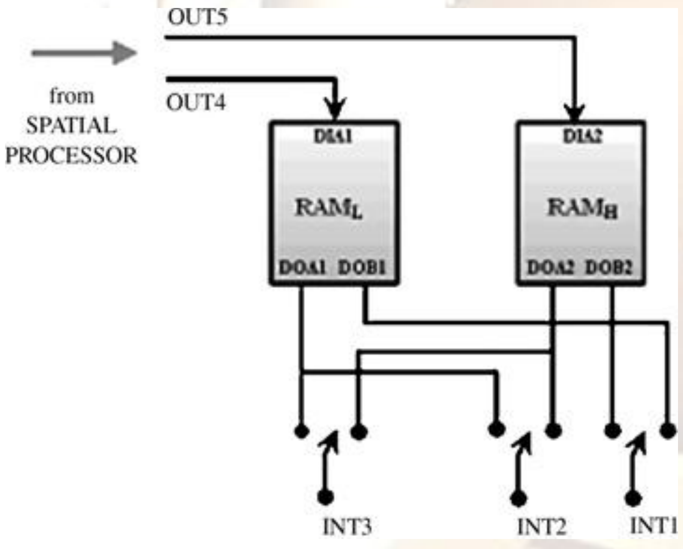


Fig. 7. Arrangement of SMEM frame buffers.

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 2*i* during computations involving slice *i* (*i* = 1, 2, 3, . . . ).

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 $\log_2(N^2)$ cycles. Fig. 9 helps us

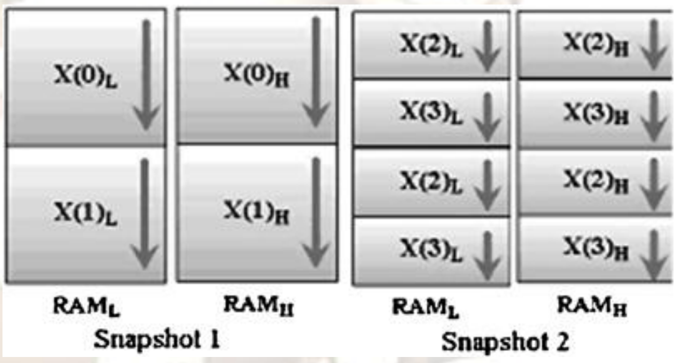
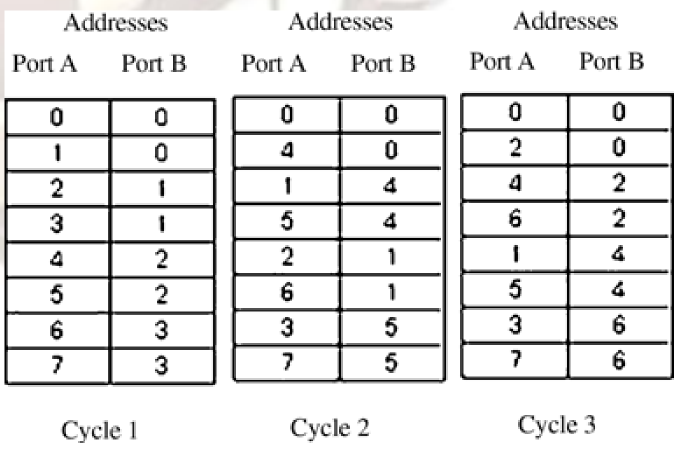


Fig. 8. Two snapshots of the SMEM.



| Addresses | | Addresses | | Addresses | |
|---|---|---|---|---|---|
| Port A | Port B | Port A | Port B | Port A | Port B |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 4 | 0 | 2 | 0 |
| 2 | 1 | 1 | 4 | 4 | 2 |
| 3 | 1 | 5 | 4 | 6 | 2 |
| 4 | 2 | 2 | 1 | 1 | 4 |
| 5 | 2 | 6 | 1 | 5 | 4 |
| 6 | 3 | 3 | 5 | 3 | 6 |
| 7 | 3 | 7 | 5 | 7 | 6 |
| Cycle 1 | | Cycle 2 | | Cycle 3 | |

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

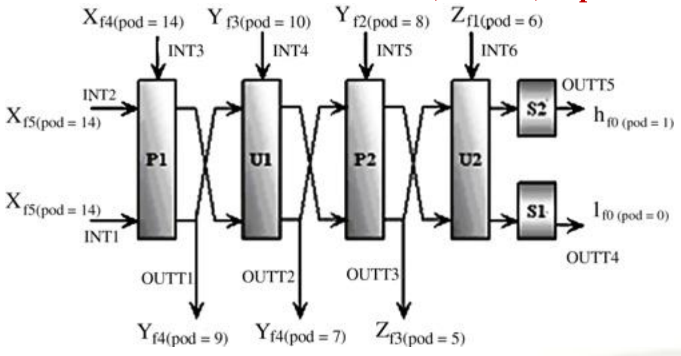


Fig. 10. Snapshot of the TP. to visualize the addressing pattern for a sample RAM depth of 8.

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. 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

$$T = 2N^2 + 2N + LatencyRPE + LatencyCPE + LatencyTP\ . \quad (6)$$

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 $5N^2 + 5N$.

## 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.

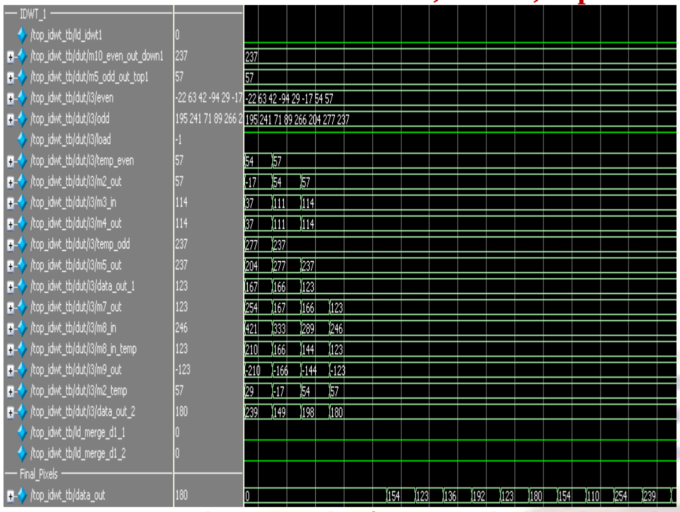
### B. Implementation Results

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, 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 | $2N2 + 2N\psi + 47$ clock cycles |
| Number of occupied slices | 1776 (2%) |
| Total number four input LUTs | 2188 (1%) |
| Number of block RAMs | 350 (63%). |

### C. Inverse discrete wavelet transform

The inverse DWT (IDWT) is the computational reverse. The lowest low-pass and highpass data-streams are up-sampled (ie. a zero is placed between each data-word) and then filtered using filters related to the decomposition filters. The two resulting streams are simply added together to form the low-pass result of the previous level of processing. This can be combined with the high-pass result in a similar fashion to produce further levels, the process continuing until the original data-stream is reconstructed.



```
IDWT_1
/top_idwt_tb/ld_idwt1                     0
/top_idwt_tb/dut/m10_even_out_down1       237                    237
/top_idwt_tb/dut/m5_odd_out_top1          57                     57
/top_idwt_tb/dut/i3/even                  -22 63 42 -94 29 -17   -22 63 42 -94 29 -17 54 57
/top_idwt_tb/dut/i3/odd                   195 241 71 89 266 2    195 241 71 89 266 204 277 237
/top_idwt_tb/dut/i3/load                  -1
/top_idwt_tb/dut/i3/temp_even             57                     54    57
/top_idwt_tb/dut/i3/m2_out                57                     -17   54    57
/top_idwt_tb/dut/i3/m3_in                 114                    37    111   114
/top_idwt_tb/dut/i3/m4_out                114                    37    111   114
/top_idwt_tb/dut/i3/temp_odd              237                    277   237
/top_idwt_tb/dut/i3/m5_out                237                    204   277   237
/top_idwt_tb/dut/i3/data_out_1            123                    167   166   123
/top_idwt_tb/dut/i3/m7_out                123                    254   167   166   123
/top_idwt_tb/dut/i3/m8_in                 246                    421   333   289   246
/top_idwt_tb/dut/i3/m8_in_temp            123                    210   166   144   123
/top_idwt_tb/dut/i3/m9_out                -123                   -210  -166  -144  -123
/top_idwt_tb/dut/i3/m2_temp               57                     29    -17   54    57
/top_idwt_tb/dut/i3/data_out_2            180                    239   149   198   180
/top_idwt_tb/ld_merge_d1_1                0
/top_idwt_tb/ld_merge_d1_2                0
Final_Pixels
/top_idwt_tb/data_out                     180                    0     154 123 136 192 123 180 154 110 254 239
```

Fig: Simulation Result of IDWT Block with Final Pixel Values

Figure shows the final pixel value of the original image calculated and hence they are given out serially. Thus the pixel values given at the input of the DWT block is compared and verified with the final output pixel vales from the IDWT block.

## 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. Bernab´e, J. Gonz´alez, J. M. Garc´ia, 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 multidimensional 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.

[24] A. Secker and D. Taubman, "Motion-compensated highly scalable video compression using an adaptive 3-D wavelet transform based on lifting," in *Proc. IEEE Int. Conf. Image Process.*, Thessaloniki, Greece, Oct. 2001, pp. 1029–1032.

[25] B. Pesquet-Popescu and V. Bottreau, "Three-dimensional lifting schemes for motion compensated video compression," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Salt Lake City, UT, May 2001, pp. 1793–1796.

[26] N. Bozinovi´c, J. Konrad, T. Andr´e, M. Antonini, and M. Barlaud, "Motion-compensated lifted wavelet video coding: Toward optimal motion/transform configuration," in *Proc. 12th Eur. Signal Process. Conf.*, Vienna, Austria, Sep. 2004, pp. 1975–1978.

[27] L. Luo, S. Li, Z. Zhuang, and Y.-Q. Zhang, "Motion compensated lifting wavelet and its application in video coding," in *Proc. IEEE Int. Conf. Multimedia Expo*, Tokyo, Japan, 2001, pp. 365–368.

[28] *Architecture and Features of a Fully Scalable Motion-Compensated 3-D Subband Codec*, document M7977.doc, ISO/IEC/JTC1/SC29/ WG11, Mar. 2002.

[29] *Improved MC-EZBC with Quarter-Pixel Motion Vectors*, document 813 MPEG2002/M8366, ISO/IEC JTC1/SC29/WG11, May 2002.