ISSN (Online): 2230-7893
www.IJCEM.org

Architecture for Three-Dimensional (3-D) Discrete Wavelet Transform for Grayscale image

Vikas Kumar[1],Sonal[2]
[1]Assistant Professor,N.C College of Engineering,Israna,
beniwal_vikas23@rediffmail.com
[2]Lecturer, School of Engg. and Sciences, B.P.S.M.V, Khanpur-Kalan,
sonalkharb@rediffmail.com

Abstract

Three-dimensional discrete wavelet transform (3-D DWT) is used for compression applications. DWT allows for better compression on 3-D data as compared with 2-D methods. This paper describe an architecture for the three-dimensional DWT, called the 3DW-I. 3DW-I architecture is based on folding. Applications of this architecture include high definition television (HDTV) and medical data compression, such as magnetic resonance imaging (MRI). Three-dimensional DWT is implemented in 3DW-I architecture like to folded one-dimensional and two-dimensional designs. 3-D DWT allows even distribution of the processing load onto 3 sets of filters, with each set performing the calculations for one dimension. The control for this design is very simple, since the data are operated on in a row–column and image fashion. The architecture is described according to memory requirements, number of clock cycles, and processing of frames per second.

Keywords: Computer Architecture, Digital Filters, Digital Signal Processors, HDTV, Wavelet Transforms.

1. Introduction

Popular application for the discrete wavelet transform (DWT) is compression. This paper introduces architecture for the 3-D DWT: the first architecture to be proposed. These are the following; variables are commonly used in wavelet architecture literature that will be used throughout this paper.

g Highpass (wavelet) filter coefficients;
h Lowpass (scaling) filter coefficients;
J Total number of octaves;
j Current octave (used as an index $1 \le j \le J$);
N Total number of inputs;
n Current input (used as an index $1 \le n \le N$).
L Width of the filter;
L1,L2, and L3filter widths for the x, y, and z dimensions, respectively;
k Current filter coefficient;
$W_{high}(j,n)$ Discrete wavelet transform of function f;
W (j, n) Approximation of the signal in the jth "scaling" subspace of , except for
W (0, n) Which is the input signal.

The pairs of filters are used for computing the wavelet transform. One filter in the pair calculates the wavelet coefficients; other filter applies the scaling function. This scaling function, which is implemented with the filter coefficients *h* and g.
The lowpass output:

$$W(j,n) = \sum_{m=0}^{L} W(j-1,m)h(2n - m) \quad (1)$$

The highpass output:

$$W\,high(j,n) = \sum_{m=0}^{L} W(j-1,m)g(2n - m) \quad (2)$$

The above equations demonstrate, the Discrete Wavelet Transform can be computed using difference equations, which finite impulse response (FIR) filters perform.
In this paper for computing the 3-D DWT are based on the folded filter, with a semi-systolic design of regular MACs and a short latency and on the block-based approach with a very small on-chip memory requirement.
The 3-D wavelet transform is shown conceptually in Figure 1, with lowpass and highpass filters, shown with a down-sampler attached to each filter. Implemented directly, it would take 14 filters, as shown in Figure 2. However, this is not an optimal solution. Instead, these filters would be implemented as polyphase ladder [10], [9], lattice [7], [15], [3], or folding [4]. The filters (see Figure 2) of the Y dimension would be active only half as often as those of the X dimension. Similarly, the Z –dimension filters would only be one quarter as busy as the X dimension filters. One way to balance the amount of work among filters is to fold the design [1], [6].
The architectures for the 3-D discrete wavelet transform are based on folded. The architecture, which is called 3DW-I, is a direct mapping of the 3-D DWT. For this design, the data are pipelined such that

one filter pair calculates all outputs for one dimension. It operates on data sequentially along row, column, and slice of image.

2. The 3–D W-I Architecture

To perform the 3-D DWT, we study the Figure 1 and converted the diagram to architecture. This requires adding RAM and a multiplexor, as shown in Figure 2, and making adjustments for folding. The folding adjustments are expanded on in what follows. The result is the 3DW-I architecture seen in Figure 3.

2.1 Design of 3DW-I

Figure 3 shows how the conceptual model of the three- dimensional could be implemented directly, assuming the filters are folded. The 3DW-I architecture uses semi-systolic filters, each containing Li MACs, which are also called processing elements (PEs).

Here, Li stand for the number of filter taps in the i^{th} dimension. Shift registers are used inside the MACs to keep partial results flowing smoothly. The number of shift registers per MAC is dependent on the dimension only and neither the input size nor the filter size. The MACs are strung together in a linear fashion; however, the input latches should be clocked twice for every clock cycle. In other words, the data are used by every other MAC. This data flow eliminates the need for unused calculations and explicit downsampling. A better way to move the data to get inherent downsampling is to send it from the even MACs to other even MACs, whereas the odd MACs send data to the next odd MAC.

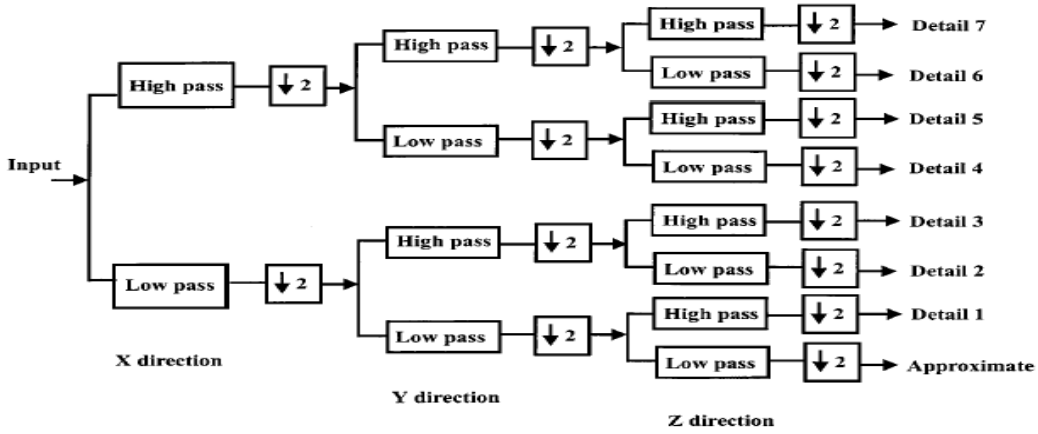

Fig. 1 Three - Dimensional Discrete Wavelet Transform(DWT)

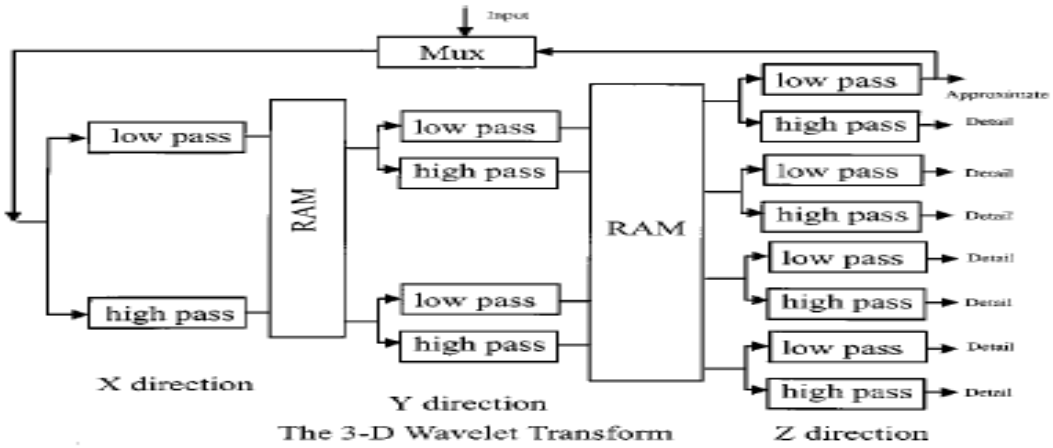

Fig. 2 Three - Dimensional Discrete Wavelet Transform (DWT) Design

ISSN (Online): 2230-7893
www.IJCEM.org

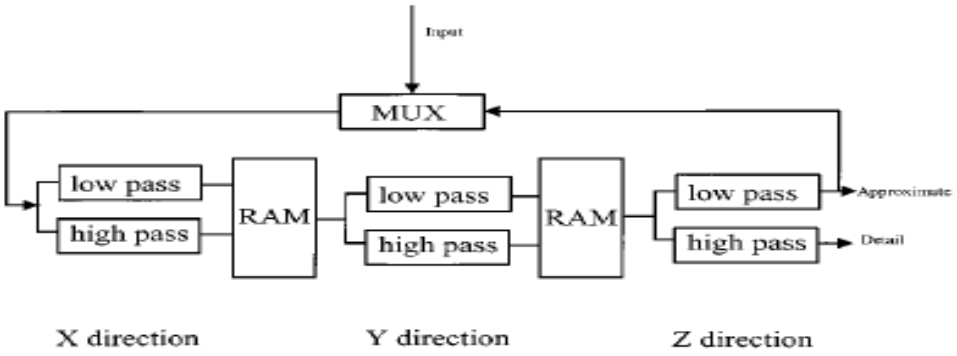

Fig. 3 Three - Dimensional DW – I Architecture

The 3DW-I architecture's folded design allows scalability to a longer wavelet. The wavelet sizes determine the number of MACs in each filter. It has simple, distributed control since each processing element has few functions:

• shift inputs
• multiply
• add

The MACs have only a few internal registers, which are consistent through the filter. In other words, any two MACs in the filter have the same number of registers. The 3DW-I is cascadable, meaning that another 3DW-I could be used to compute the next octave of resolution, like other folded designs. Finally, the serial filters generate results in a low number of clock cycles.

In the 3DW-I architecture, the amount of memory between filters is a concern. To get started in the Y direction, the X direction must generate enough results: one row of outputs for every filter coefficient in the Y dimension. Similarly, the Z filters must wait on the Y dimension filters to finish enough outputs for multiple images. To keep the data flowing, it must be buffered correctly. Otherwise, one filter's output will overwrite a previous output, or the next filter set will attempt a data read before the data is there. Obviously, for even a moderate size data volume, filters for all three dimensions will be active most of the time. During the final octave's computations, the X filters will finish first, and then, the Y filters will be followed by the Z filters. This analysis is expanded upon in what follows.

2.2 The 3DW-I Time Delay

In the 3DW-I, each MAC will do the following in one clock cycle:

t1 data shift and a partial calculation shift (done in parallel).
t2 one multiplication;
t3 one addition;

Thus, the minimum clock cycle time for the 3DW-I architecture is determined by t1 + t2 + t3.

For the 3DW-I architecture, there are (T1 + T2 + T3) multiplications per clock cycle since there are L1 multipliers per X filter, L2 multipliers per Y filter, L3 multipliers per Z filter, and each dimension has two filters. There are additions per clock cycle. The number of clock cycles is determined by the input dimensions, as well as the lower octave scheduling. only single octave results are used. The number of clock cycles for the 3DW-I design depends on the input dimensions. The number of inputs, of course, is the height, width, depth, or NMP. Each filter pair NMP/2 needs clock cycles to complete its operations. However, there is a delay before the Y and Z filters can start. The Y filters must wait until the X filters have generated N (M/2) outputs due to the fact that they sample data along columns instead of rows. Similarly, the Z filters must wait for (N / 2) (M / 2) P outputs from the Y filters since they sample data along the slices. Thus, the Z filters cannot start until (N / 2) (M / 2) P + N (M / 2) clock cycles. The Z filters finish at time (N / 2)(M / 2) P + NM / 2 + NMP / 2 .The term (L1 + L2 + L3) / 2 should be added to this to get the exact number of clock cycles since it takes the depths of the filters into account. However, this term is compared with the overall time needed.

3DW-I Total clock cycles for octave 1

$$= (N / 2)(M / 2)P + NM / 2 + NMP / 2 + (L1 + L2 + L3) / 2$$

For the second octave, the above equation also holds, except that the input size has been reduced by a factor of 2 in each dimension. i.e. replace N with N / 2, M with M / 2, and P with P / 2.

3DW-I Total clock cycles for octave 2

$$= (N / 2^2)(M /2^2) P / 2 + (N / 2) M / 2^2 + (N / 2)(M / 2) P / 2^2 + (L1 + L2 + L3) / 2$$

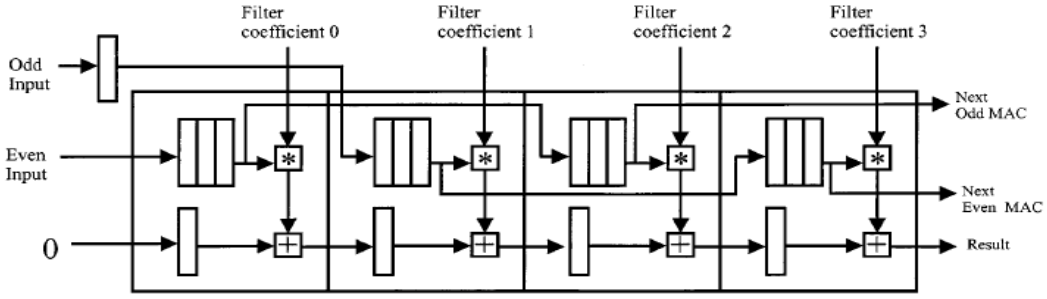

Fig. 4 Medium Access Control Structure with Registers

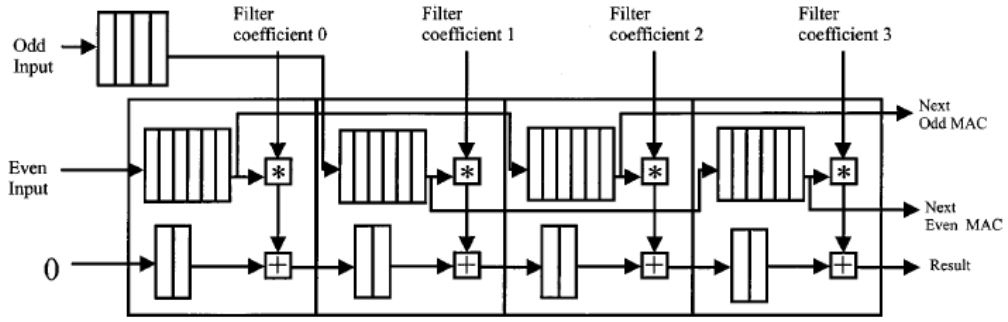

Fig. 5 Revised Medium Access Control Structure for folding

For octave j, the amount of computations is

3DW-I Total clock cycles for octave j

$$= 3NMP / 2^{3j-1} + NM / 2^{2j-1} + (L1 + L2 + L3) / 2$$

To find the total amount of clock cycles to compute all octaves

3DW-I Total clock cycles for all octaves

$$= \sum_{j=1}^{J} \left(\frac{3NMP}{2^{3j-1}} + \frac{NM}{2^{2j-1}} + \frac{L1+L2+L3}{2} \right)$$

For multiple octaves, there are two possibilities. One is that the architecture schedules other calculations for other octaves between calculations for the first octave, such as is demonstrated for the 1-D transform in [8] and echoed for the 2-D transform in [2]. The problem is that for the 3DW-I, which is based on the same ideas as [8] and [2], this scheduling is not optimal. Here is why: Scheduling every other clock cycle to the first octave means that the time necessary to compute the whole transforms (all octaves) is twice the number of clock cycles for the first octave. However, this will always be greater than the sum of the number of clock cycles needed for all octaves. i.e.

$$2\left(\frac{3NMP}{4} + \frac{NM}{2} + \frac{NMP}{2} + c\right) >$$

$$\sum_{j=1}^{J} \frac{3NMP}{2^{3j-1}} + \sum_{j=1}^{J} \frac{NM}{2^{2j-1}} + cJ$$

Taking the limits of the summations as J → ∞, then we will see

$$\left(\frac{3NMP}{2} + NM + 2c\right) > \left(\frac{6NMP}{7} + \frac{2NM}{3} + cJ\right)$$

Since 3/2 > 6/7, and 1 > 2/3, the only possible way to make the above inequality false is by having a value of J such that cJ is larger than, say, 9NMP /14 + NM/3 . However, the maximum value for J = $\min(\log_2 N, \log_2 M, \log_2 p)$. and therefore, there is no way that it can cause the above inequality to be false. Therefore, interleaving the lower octave calculations

ISSN (Online): 2230-7893
www.IJCEM.org

will be less efficient for the 3-D case than using the transformers in a cascaded fashion, that is, where the coefficients for octave j are only performed after all coefficients for octave j + 1 have been computed. This is the reason why only the first octave is considered when comparing the 3-D architectures.

2.3 The 3DW-I Memory Requirements

The Y inputs depend on X the outputs [26], whereas Z the inputs depend on Y the outputs. Thus, the operations cannot be rearranged. The X filters take their inputs in a column, row, and slice (image) fashion. The Y filters then take their inputs in a row, column, and slice fashion. Then, the Z filters go along the slices first. If X reads NMP inputs, then two (conceptual)

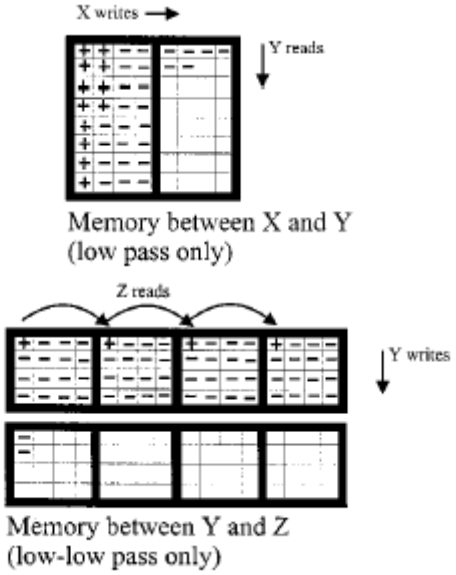

Figure 6. Random Access Memory of Three-DW-I

Y filters will read NMP/2 inputs. Then, four (conceptual) Z filters will read NMP/4 inputs. Each Y filter must wait until NM/2 inputs are available, whereas each Z filter must wait until (N/2)(M/2) P of its inputs are available. This process is shown in Figure 6, where results from the X dimension's filters are marked with a horizontal dash. The vertical dashes indicate that these locations have been read by the Y dimensional filters and can be reused. The Y filters will get two values from the low pass memory on even clock cycles and then two values from the highpass memory on odd clock cycles. The reason Y filters get two values per clock cycle is the same as for the X filters: the inherent downsampling operation. One value is passed to the first even MAC, whereas the other value goes to the first odd MAC. The memory between the Y and Z filters is four sets of NMP/4 or NMP. Therefore, the design depends on the input data's dimensions. For example, assume a small input size of a four -frame sequence of 8 × 8 images. Each filter downsamples the outputs and writes to two memories. Since the original data size is 8 × 8 × 4, the data size reaching one of the Y filter pairs is 8 × 4 × 4 due to the downsampling operation. Note that the filters writing to this memory store one value per clock cycle, but the filters reading the memory will fetch two values at a time. The total amount of memory between the X and Y filters is 2NM , whereas the amount of memory required between Y and Z filters is (M/2)(N/2) P(2)(4) or 2NMP . There must be four sets of the memory: one for each filter output stream. There also must be two buffers for each stream: one to keep the data until it is ready to use and another to collect data until the first set has been completely read. Thus, the total memory for the 3DW-I is given as

Total memory needed by DW-I

$$= (2NMP + 2NM + 2(L1 + L2 + L3))\text{Precision}$$

3. Performance Analysis

This section show the performance of the 3DW-I architectures. The subsections deal with memory requirements, time delay, and overall speed. Only one octave is considered in this architectures work best when octaves are performed one at a time. Performing a second level of resolution takes a proportional amount of resources (time and memory), except that the data is reduced by a factor of 2 along each dimension, i.e. $N_2 = N/2$, $M_2 = M/2$, and $P_2 = P/2$. Therefore, for comparison, only the first octave is considered.

This architecture is designed with sets of grayscale images in mind. However, 3DW-I could be expanded to color video. A simple solution would be to treat color video as three different signals and use this architecture on these signals separately. Another possibility is to use each red–green–blue value per pixel as a single value and perform the 3-D DWT on this signal. Again, either architecture could work on this signal, although the internal bus widths would have to be wide enough to accommodate this data (e.g., 24 bits for the input path).

3.1 Memory Requirement

Figure 7 shows the on-chip memory requirement for the architectures, which can be expressed as

DW-I Memory for octave
= (2NMP + 2NM + 2(L1 + L2 + L3)) words

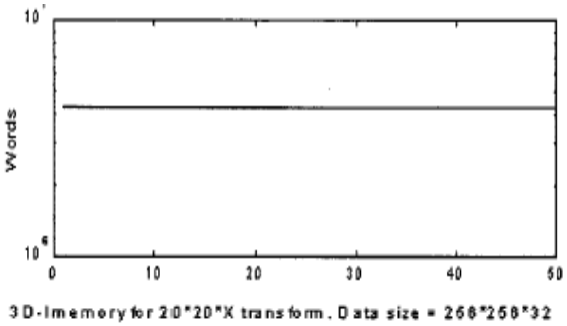

Fig. 7 On-chip memory requirements versus data size

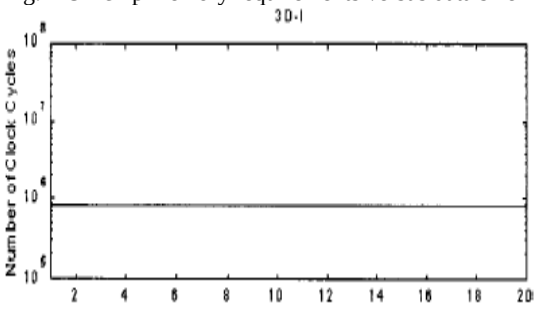

Fig. 8 Time requirement (in clock cycles) for filter size L_3.

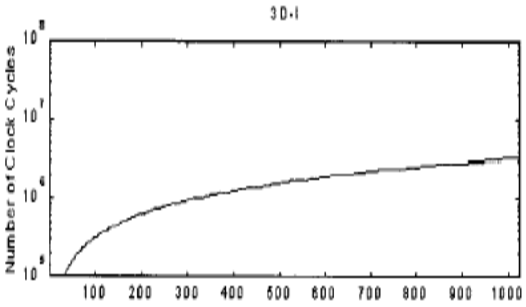

Fig. 9 Time requirement (in clock cycles) for width of images

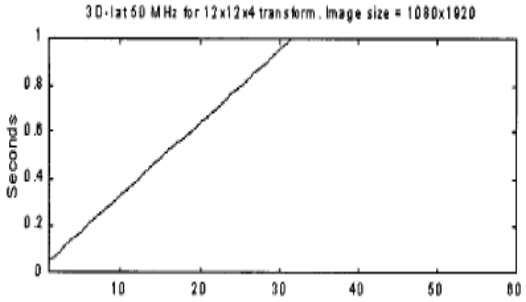

Fig. 10 Frames – per – second analysis for 12 × 12 × 4 transform and 1080 × 1920 image size

The word width equals the amount of precision used (e.g., 16 bits). As Figure 7 shows, changing the data size has a direct effect on the amount of memory needed by 3DW-I. When the wavelets increase in size, the memory requirements of the architecture change. The values L1 and L2 are chosen to be large if not impractical. Both of these parameters correspond to a filter of 20 taps, although wavelets used seldom have support greater than 12 taps [25]. In fact, simulating the memory needs with L1 and L2 values of 100 taps each, the wavelet corresponding to L3 would have a width of 425 taps would surpass the memory needed by 3DW-I. The other extreme, which is to equalize the memories, is to make the data size trivial, which is not realistic. Either way, the 3DW-I architecture will always need several magnitudes more of on-chip memory.

3.2 Time Delay

The analysis of the time delay for architecture produced the following two equations, based on the data's dimensions (*NMP*) and the size of the wavelet filters (L1, L2 L3):

3 DW-I Total clock cycles for octave
$$= (N / 2)(M / 2)P + NM / 2 + NMP / 2 + (L1 + L2 + L3) / 2$$

TABLE 1
MINIMUM CLOCK SPEED (IN MEGAHERTZ) FOR 3DW-I WITH A 4× 4 × 2 WAVELET

Image Size:	256×256		512×512		720×1280		1080×1920	
FPS:	30	60	30	60	30	60	30	60
3DW-I MHz	2	3	6	12	21	42	48	94

TABLE 2
MINIMUM CLOCK SPEED (IN MEGAHERTZ) FOR 3DW-I WITH A 12 × 12 × 4 WAVELET

Image Size:	256×256		512×512		720×1280		1080×1920	
FPS:	30	60	30	60	30	60	30	60
3DW-I MHz	2	3	6	12	21	42	48	94

The graphs seen in Figures 11 and 12 shows how the two architectures compare with regards to number of lock cycles. The number of multiplications and number of additions directly depend on the clock cycles. In Figure 8, varying the size of the wavelet has little impact on 3DW-I. Figure 9 shows the clock cycles of 3DW-I architecture,

3.3 Speed

Current 1-D DWT processing chips have speeds ranging from 5 MHz [11] to 30 MHz [14]. At an achievable speed of 200 MHz (see [13]), the architecture would be able to perform one octave of the 3-D DWT on a sequence of images without a

problem. A large wavelet transform of 12 × 12 × 4 was chosen for the purpose of comparing the architectures. Although MRI data (256 × 256) was used with the simulation, the image size of 1080 × 1290 in Figure 10 was chosen to be compatible with HDTV. The graphs of Figure 10 show how fast the chips would need to run to guarantee 30 f/s. However, the 3DW-I architecture is able to perform at even much slower clock frequencies, as the graph shows. This question is not easy to answer since the frames per second depend on both image size and filter lengths Tables 1 and 2 lists the lowest operating frequency for the architecture to achieve 30 and 60 f/s. Remember that these results are based on one of each chip. These tables come from the frames-per-second equations

$$3\text{ DW-I fps} = (4(\text{Speed}) - 2NM - 2(L1 + L2 + L3)) / (NM + 2NM)$$

The problems of the architecture are memory requirements and latency. It requires words for intermediate data storage. Whereas this may be reasonable to fit on one chip for small images, the design would be much better if it did not require large internal memory. It would be best to have the memory independent of the input size. Finally, the latency also depends on the input size. The filters cannot start until clock cycles, which mean it takes clock cycles to generate the first output. After that, it generates an output every clock cycle. These problems of large internal memory and latency are inherent to multidimensional (semi-)systolic architectures.

4. Conclusions

The DWT can be used on 1-D, 2-D, and 3-D signals. The DWT represents an input signal as one approximate signal and a number of detail signals. The representing signals combined together need no more storage space than the original signal. These signals can be sent to the synthesis procedure to recreate the original signal without loss of information, assuming that no lossy compression is performed. Alternately, the analysis output signals can be compressed, stored, and uncompressed before being sent to the synthesis procedure. This allows the signal to be stored without losing much information. We present, in this work, architecture to accomplish the 3-D DWT, the 3DW-I, which were simulated to verify their correctness.

TABLE 3
DETAILS FOR THE 3DW-I ARCHITECTURE

	3 DW – I
Memory	Very large based on input
Cascadable	yes
Run in Parallel	Not Easily
Size	Large
Scalable to a Large Wavelet	Yes
Design	Folded
Control	Simple, Distributed
Type of Filters	Serial
Number of MACS	2(L1 + L2 + L3)
Number of clock Cycles	Low

The 3DW-I architecture is a straightforward implementation of the 3-D DWT. It allows even distribution of the processing load onto three sets of filters, with each set doing the calculations for one dimension. The filters are easily scalable to a larger size. The control for this design is very simple and distributed since the data are operated on in a row–column-slice fashion. The design is cascadable, meaning that the approximate signal can be fed back directly to the input to generate more octaves of resolution. Scheduling every other time slot to do a lower octave computation works for this design. The filters are folded, allowing the multiple filters in the dimensions to map onto a single pair for each dimension. Due to pipelining, all filters are utilized 100% of the time, except for the start-up and wind-down times.

In conclusion, this paper presents architecture for the 3-D DWT. Three-dimensional data, such as medical applications, need a true 3-D transform, and this is the architecture to perform the 3-D DWT. Architecture has advantages and disadvantages such as memory requirements, latency, and frames per second. Therefore, architecture is used depends on the application parameters. For example, a television application has fixed dimensions on the data and needs the results as fast as possible. Therefore, the 3DW-I architecture suits it well.

5. Simulation Results

Simulation Results for the 3DW-I Architecture

The results that follow were obtained after performing the 3-D DWT on a 256 × 256 × 32 grey-scale MRI data set with the 3DW-I simulation. It used Daubechies 8 coefficients for the X and Y dimensions, followed by the two Haar coefficients in the Z dimension.

X dimension with Daub 8 coefficients

Start time = 0
Image height = 256 Image width = 256
Number of images = 32 Input width = 8
X dir end time = 1048581
Number of mults 16 777 312
Number of adds 16 777 312
Data read 2 097 152

Y dimension with Daub 8 coefficients (2 sets of RAM)

Start time = 3276800
Image height = 256 Image width = 128
Number of images = 32 Input width = 16
Y dir end time = 32768 + 1048584
= 1081352
Number of mults 16 777 376
Number of adds 16 777 376
Data read 20 97 152 × 2

Z simulation with Haar coefficients (4 sets of RAM)

Start time = 32768 + 524288 = 557056
Image height = 128 Image width = 128
Number of images = 32 Input width = 16
Z dir: end time = 557056 = 1048592
= 1605648
Number of mults 41 94 384
Number of adds 4 194 384
Data read 2 097 152.

References

[1] M. Weeks and M. Bayoumi, "3-D discrete wavelet transform architectures,"in *Proc. IEEE Int. Symp. Circuits Syst.*, Monterey, CA, May31–June 3 1998.
[2] M. Vishwanath, R. Owens, and M. Irwin, "VLSI architectures for thediscrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 42, pp.305–316, May 1995.
[3] P. Rieder, J. Götze, and J. A. Nossek, "Implementation of discrete vectorvaluedwavelet transforms," in *Proc. IEEE Int. Symp. Circuits Syst.*,Hong Kong, June 9–12, 1997, pp. 369–372.
[4] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete waveletransforms," *IEEE Trans. VLSI Syst.* , vol. 1, pp. 191–202, Apr. 1993.
[5] Michael Week, Magdy A. Bayoumi "A Three dimensional Discrete Wavelet Transform Architectures" in *Proc. ieee*, 8 Aug. , 2002, vol.50.
[6] G. Zhang, M. Talley, W. Badawy, M. Weeks, and M. Bayoumi, "A low power prototype for a 3-D discrete wavelet transform processor," in *Proc. IEEE Int. Symp. Circuits Syst.*, Orlando, FL, May 30–June 2 1999.
[7] T. C. Denk and K. K. Parhi, "Architectures for lattice structure based orthonormal discrete wavelet transforms," in *Proc. IEEE Int. Conf. Applicat.Specific Array Process.*, Aug. 1994, pp. 259–270.
[8] "On the synthesis of regular VLSI architectures for the 1-D discretewavelet transform," *Proc. SPIE*, vol. 2303, pp. 91–104, July 24–29, 1994.
[9] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1997.
[10] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Applicat.*, vol. 4, no. 3, pp. 245–267,1998.
[11] G. Gonzalez-Altamirano, A. Diaz-Sanchez, and J. Ramirez-Angulo, "Fast sampled-data wavelet transform CMOS VLSI implementation," in *Proc. 39th Midwest Symp. Circuits Syst.*. Ames, IA: Iowa State Univ., August 18–21, 1996, pp. 101–104.
[12] "Discrete wavelet transform: Data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. Signal Processing*, vol. 45, pp. 1291–1308, May 1997.
[13] "Pentium® processor family developer's manual," Intel Corp., Hillsboro, OR, 1997.
[14] "Wavelet transform processor chip user's guide," Aware, Inc., Bedford, MA, 1994.
[15] "VLSI architectures for lattice structure based orthonormal discrete wavelet transforms," *IEEE Trans. Circuits Syst. II*, vol. 44, pp.129–132, Feb. 1997.
[16] W. W. Boles and Q. M. Tieng, "Recognition of 2-D objects from the wavelet transform zero-crossing representation," *Proc. SPIE*, vol. 2034, pp. 104–114, July 11–16, 1993.
[17] C. M. Brislawn, J. N. Bradley, R. J. Onyshczak, and T Hopper, "The FBI compression standard for digitized fingerprint images," *Proc. SPIE*, vol. 2847, pp. 344–355, Aug. 4–9, 1996.
[18] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, pp. 759–771, Mar. 1995.
[19] C. Chakrabarti, M. Vishwanath, and R. Owens, "Architectures for wavelet transforms: A survey," *J. VLSI Signal Process.*, vol. 14, no. 1, pp. 171–192, 1996.
[20] C. Chakrabarti, "A DWT-based encoder architecture for symmetrically extended images," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. IV, Orlando, FL, May 30–June 2, 1999, pp. 123–126.
[21] "Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing," in *Proc. Int. Symp. Circuits Syst.*, 1994, pp. 77–81.
[22] T. Edwards, "Discrete wavelet transforms: Theory and implementation," Stanford Univ., Stanford, CA, unpublished paper, June 4, 1992.
[23] J. Fridman and E. S. Manolakos, "Distributed memory and control VLSI architectures for the 1-D discrete wavelet transform," in *Proc. IEEE VLSI Signal Process. VII*, La Jolla, CA, Oct. 26–28, 1994, pp. 388–397.
[24] K. H. Goh, J. J. Soraghan, and T. S. Durrani, "New 3-D wavelet transform coding algorithm for image sequences," *Electron. Lett.*, vol. 29, no. 4, pp. 401–402, 1993.
[25] "Discrete wavelet transform: Data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. Signal Processing*, vol. 45, pp. 1291–1308, May 1997.
[26] M. A. Trenas, J. López, and E. L. Zapata, "A memory system supporting the efficient SIMD computation of the two dimensional DWT," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 3, 1998, pp. 1521–1524.