ARCHITECTURES FOR THE 3-D DISCRETE WAVELET TRANSFORM

A Dissertation

Presented to

The Graduate Faculty of

The University of Southwestern Louisiana

In Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Michael Clark Weeks

Spring 1998

ARCHITECTURES FOR THE 3-D DISCRETE WAVELET TRANSFORM

Michael Weeks

APPROVED:

_____	_____
Magdy A. Bayoumi, Chair Edmiston Professor of Computer Engineering	C. Henry Chu Associate Professor of Computer Engineering
_____	_____
Kimon Valavanis Professor of Computer Engineering	Guna Seetharaman Associate Professor of Computer Engineering
_____	_____
Gui-Liang Feng Associate Professor of Computer Engineering	Lewis Pyenson Dean, Graduate School

ACKNOWLEDGMENTS

The author would like to recognize the following people for their help in this work: Dr. Magdy Bayoumi, for serving as advisor and chair of the dissertation committee; Dr. Henry Chu who served on the committee and provided the sample MRI data; Dr. Kimon Valavanis, Dr. Guna Seetharaman, and Dr. Gui-Liang Feng who served on the committee; and Dr. Jimmy Limqueco and Dr. Paul Shipley for their useful discussions.

The author expresses appreciation for funding from grant LEQSF(1996-99)-RD-B-13, "Interoperable Underground Wavelet Transients Munitions Classification".

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This work deals with the Wavelet Transform (WT), and more specifically, the Discrete Wavelet Transform (DWT). The goal of this dissertation is to present the need for a 3-D Discrete Wavelet Transformer, review designs for the DWT, and propose 2 architectures for the 3-D DWT. The designs reviewed for the DWT are for the 1 and 2-dimensional cases, since this is one of the first works to address the 3-D case.

1.1 WHAT ARE WAVELETS?

What is a wavelet, and what does it look like? "The DWT is a basis transformation, i.e., it calculates the coordinates of a data vector (signal or spectrum) in the so-called wavelet spectrum. A wavelet is a function that looks like a small wave, a ripple of the baseline, hence its name" [Walczak96].

The Discrete Wavelet Transform (DWT) gets its name from being discrete-time and discrete-scale. In other words, the DWT coefficients may have real (floating-point) values, but the time and scale values used to index these coefficients are integers. In the DWT, a subband filter transforms a discrete-time signal into an average signal and a detail signal. Figure 1.1 shows this concept for 1 octave, where an octave is a level of resolution. The figure includes the analysis (wavelet transform) on the left side and the synthesis (inverse wavelet transform) on the right side. A low-pass filter produces the average signal, while a high-pass filter produces the detail signal. Multiresolution analysis [Mallat89] feeds the average signal into another set of filters, which produces the average and detail signals at the next octave. The detail signals are kept, but the higher octave averages can be discarded, since they can be computed during the synthesis. Each octave's

outputs have only half the input's amount of data. Thus, the wavelet representation is the same size as the original. The multiresolution analysis for 3 octaves appears in figure 1.2.

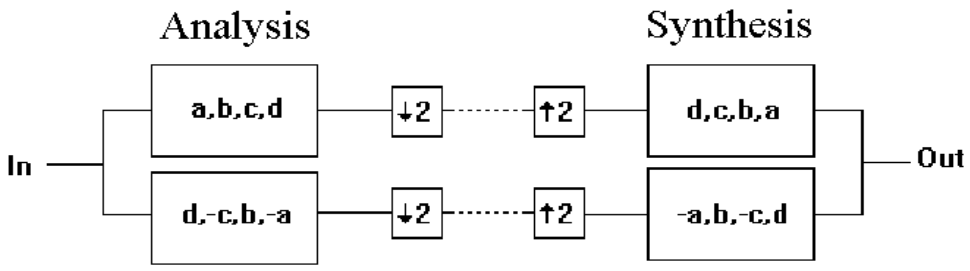


Figure 1.1 - A 1-octave, 4 tap DWT

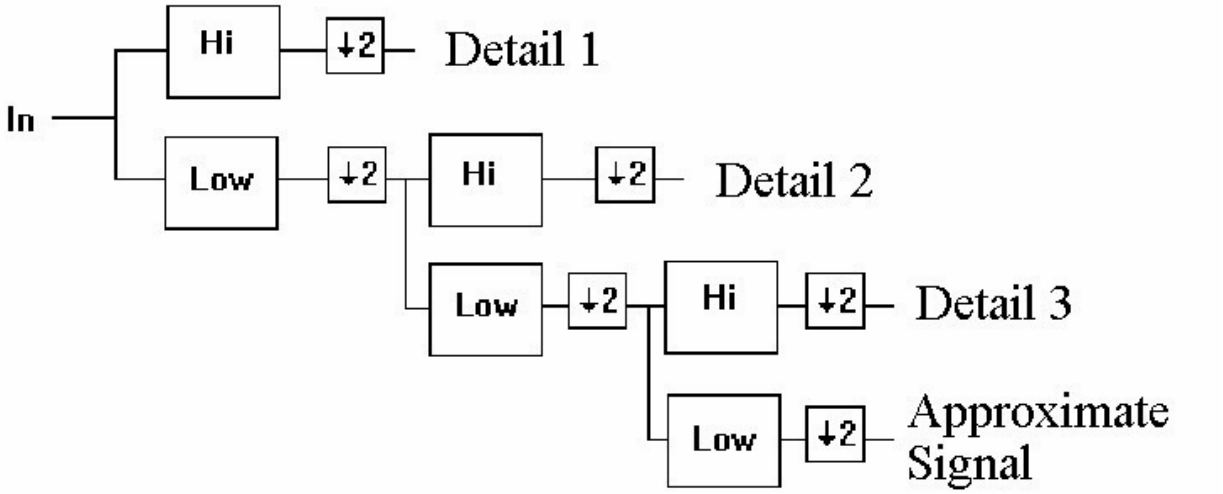


Figure 1.2 - Three level multiresolution

Traditional signal analysis and image processing applications use Fourier methods such as the Fourier Transform and the Discrete Cosine Transform. However, wavelets handle a signal's

discontinuities better than Fourier methods. Since most real-world signals have discontinuities and are of finite length, the wavelet transform better suits them. For image data, the DWT represents the image in a way that reduces or eliminates redundancy within the signal. Therefore, the signal can be compressed and stored in less space after the transform than it could before the transform. This space savings is the goal of compression, since it can be translated directly into cost. For example, transmitting a 100 Mb file from one computer to another using modems with a transfer rate of 32.2 Kb/s takes 10240/3 sec, or approximately 50 minutes. If the computer user is being charged for a long-distance phone call, the file transfer may cost $10. A modest compression of 90% the original size still saves the sender $1. In fact, Compaq is developing a wavelet-based real-time compression system for Internet communications [Hickman97].

The most significant use for the DWT so far has been in compression. "The perfect reconstruction property of the analysis and synthesis wavelets and the absence of perceptual degradation at the block boundaries favor use of wavelets in video coding applications" [Parhi93]. For example, Acharya [Acharya97] specifies a pipelined VLSI architecture for adaptive digital image compression. The compression is done by a DWT of the input, followed by spectral classification, followed by the actual compression algorithm. Compression has three steps. The first step is the transform, with which this work is primarily concerned. The transform step represents the data in a different form, but there is no information lost. The second step, quantization, maps data values to a finite set, which loses information. The name “lossy” compression comes from quantization. The third step encodes the data in a more compact way. For example, the sequence 11000000 would be encoded much like a person would say it: as two “1”s followed by six “0”s.

1.2 WAVELETS vs. FOURIER

Jean Baptiste Joseph Fourier came up with the idea of using sinusoids as bases for signal decomposition in 1822. Since then, his analytic method has been used in many signal processing applications in all fields of science and engineering. While it is still useful today, it suffers from some problems that the wavelet transform does not. It assumes that the entire signal is known for all possible time values, and that the signal is periodic. Real world signals do not fit these criteria. To deal with these shortcomings, the Windowed Fourier Transform was developed. The Windowed FT and its variations, such as the Discrete Cosine Transform, are predecessors of the Wavelet Transform.

Wavelets are used to analyze signals for which the Fourier Transform is inadequate. The Fourier Transform is good for sinusoidal signals, like a voice pattern. But the Fourier Transform has difficulty with discontinuous signals, like an image with edges. Both transforms break down a signal into simpler signals, called sinusoids for the Fourier Transform, and wavelets for the wavelet transform. These simpler signals can be added together to recreate the original. One difference between wavelets and Fourier Transforms is that the wavelet exists for a few cycles, while the sinusoid cycles forever. This limited duration means that the wavelet transform is more efficient. For example, Bruce et al. [Bruce96] shows a saw-tooth pattern broken down by a wavelet into 16 signals. A Fourier Transform of the same signal would require 256 sinusoids, since the sharp drop in the saw-tooth is difficult for the Fourier method to represent.

Wavelet analysis applies to many different fields, such as "digital communications, remote sensing, biomedical signal processing, medical imaging, astronomy and numerical analysis" [Bruce96], as well as digital signal compression, and noise reduction. One limitation of wavelets is the floating-point arithmetic used which may be unsuitable for some real-time applications. As the

saw tooth example points out, the wavelets easily show edges in the analyzed signal. A casual observer would notice the discontinuity just by viewing the wavelets. The edges are not readily visible when sinusoids from a Fourier Transform of the same input signal are viewed.

A pair of waveforms, the wavelet function denoted by Ψ, and the scaling function denoted by Φ, decompose a signal into wavelets. The waveforms used depend on the application. The analysis procedure moves and stretches the waveforms to make wavelets at different shifts (starting times) and scales (durations). The resulting wavelets include coarse-scale ones that have a long duration, and fine-scale ones that last only a short amount of time.

1.3 JPEG AND MPEG STANDARDS

The Joint Picture Experts Group (JPEG) and Motion Picture Experts Group (MPEG) were formed to invent standards for compressing still images and video sequences. These standards typically use Fourier analysis.

Currently, the Moving Pictures Expert Group (MPEG) standards are used for video compression. MPEG is a subgroup of the International Standards Organization (ISO). The MPEG group makes the standards with applications in mind. Table 1.1 below shows some of the MPEG standards. Note that MPEG-6 and MPEG-8 have also been defined for wireless transmission and 4-D object description, respectively [Szu97].

Standard	Bit-rate	Application
MPEG-1	1.5 Mbit/s	CD-ROM quality video
MPEG-2	4 to 9 Mbit/s	broadcast TV video
MPEG-3	20 to 40 Mbit/s	High Definition Television (HDTV)
MPEG-4	4.8 to 64 Kbit/s	very low bit-rate (e.g. video phone)

Table 1.1 - MPEG Standards

In an MPEG standard, motion compensation predicts what the next frame will be, based on the current frame. The prediction minus the actual frame generates the difference signal. This error in prediction undergoes the DCT, which is then quantized and encoded [Szu97].

The International Telecom Union (ITU) developed H.261 and H.263 for teleconferencing systems. A teleconference has less motion than other video applications, like television. Therefore, standards for a teleconferencing system can be relaxed. The H.261 and H.263 standards limit motion vectors to +/-15 pixels [Szu97].

The goal of lossy compression, such as the Joint Photographic Experts Group (JPEG) standard, is to minimize the space needed to represent a signal. The minimized space means that the information takes fewer bytes to store, and a sender can transmit the information in less time. The JPEG standard relies on the discrete cosine transform, which does not perform as well as the wavelet transform. The wavelet transform typically produces a sparser output. A lossy compression algorithm performs a transform on the input signal, quantizes the output, then encodes the data, for example with Huffman coding. This is called transform encoding. The sparser the transform output is, the more compact the encoding will be. Therefore, the wavelet transform allows for more compression than the JPEG standard.

The Discrete Cosine Transform (DCT) is a variation of Fourier analysis in that it uses local

information about a signal to transform it. The DCT introduces blocking effects into images due to the nature of the algorithm. Fourier analysis assumes that the signal is defined for all possible time values. In other words, the signal should extend indefinitely in both positive and negative time directions. The DCT instead splits the signal up into sections, or blocks in the case of 2-D images. When the images are compressed with DCT methods, the resulting uncompressed image looks like a quilt, where the image appears "patchy" to a human observer. The Discrete Wavelet Transform does not introduce blocking since the entire image is transformed, and the low-frequency content is preserved, such as background information. "Low-frequency" gets its name from varying little in the image, versus "high-frequency" or rapidly changing information. A 3-D DWT will outperform MPEG standards due to the fact that the 2-D advantages of the DWT over the DCT will also apply to the time dimension.

1.4 WAVELET APPLICATIONS

The DWT has many applications. For 1-D data, it was shown to de-noise Nuclear Magnetic Resonance data [Bruce96]. The FBI used it to compress fingerprints, a 2-D application [Brislawn96]. Medical images are three-dimensional, and get the maximum compression from 3-D transforms [Wang95].

Wavelets "describe data in terms of frequency and time" [Henricks96]. JPEG and MPEG formats compress images and video in a similar fashion to wavelets, but wavelets are more efficient. This greater efficiency will allow PCs to have full-screen video. Other applications include medical imaging, cable TV, digital VCRs, and video-conferencing [Henricks96].

Adaptive wavelet analysis, which uses a library of wavelet shapes, has been used to remove

noise from a recording of the "First Hungarian Dance" played by Johannes Brahms. Coifman, who removed the noise from the recording [Bruce96], also works with distinguishing objects scanned by military radar [Gibbs96]. Another application, carried out by Wickerhauser, has wavelets finding cancer [Gibbs96] in mammogram images. Other medical applications include monitoring of fetal heartbeats, ultrasounds, and analyzing electrocardiograms. One massive problem of medical images is the volume of data produced, so medical images must be compressed. The focus of this work is for 3-D MRI data, so this topic will be expanded on below. Compression is the largest application for wavelets. In order to make digital broadcasts affordable, compression will no doubt be used with television programming. The company Aware from Cambridge, Massachusetts, developed a wavelet chip with compression in mind [Aware94][Gibbs96].

Higher compression rates achieved using wavelets allow for faster transmission speeds and better images in applications like CD-ROM, interactive TV, library archives, the Internet, and telemedicine. Currently, the JPEG and MPEG standards use the Discrete Cosine Transform to compress images and video. MPEG encoders take much more processing than the decoders, which results in MPEG encoders being much more expensive than MPEG decoders ($5000 versus $70). Wavelets, on the other hand, use about the same amount of processing to compress or decompress data. Analog developed a wavelet chip that encodes and decodes data. It should cost about $50, and be available in 1996. Microsoft and Intel also plan to market wavelet products [Chinnock95].

Another application of wavelets is in matrix computation. Gregory Beylkin came up with the idea of treating matrices as images [Bruce96]. For example, to quickly find the product of two matrices, the wavelet transform operates on the matrices, then the wavelet coefficients are multiplied, and the results go through the inverse transform. The researchers named this procedure after themselves, the fast Beylkin-Coifman-Rokhlin (B-C-R) solver. The approximate solution takes

much less time to calculate than the exact solution.

Noise removal can be done better with wavelet transforms since it does not automatically eliminate the high-pass signals. Instead, the process computes the wavelet transform of a signal, and removes only the coefficients below a given threshold level. The reconstructed signal thus does not contain noise, with a minimum of high-frequency signal loss. For example, large spikes in the signal will not be removed by this procedure.

Geophysical signals benefit from wavelets in two ways: noise removal, and compression. Noise removal is an obvious advantage. Compression is also very important since a seismic survey can create 3 terabytes of data. Geophysicists were able to get compression rates of up to 100:1 [Bruce96]. In a field test, geophysicists at sea compressed and transmitted seismic survey data over a satellite to a ground-based station in a few hours. They estimated that uncompressed transmission would have taken weeks, if not longer.

The FBI invented their own fingerprint compression technique, called the "wavelet/scalar quantization greyscale fingerprint compression standard". A mathematician named Chris Brislawn, from the New Mexico Los Alamos National Laboratories, wrote this standard, which achieves a 20:1 compression ratio [Brislawn96].

A final application for the wavelet transform is 2-D object recognition. For example, a system to automatically identify objects on the ground from an aerial photograph. Boles and Tieng [Boles93] present such a method for finding 2-Dimensional objects within a 2-D image. First, the algorithm represents the object's boundary by a one-dimensional signal, then performs a 1-D DWT on it. Sharp variation points within the transformed signal are called zero-crossings. The algorithm successfully recognizes objects based on the boundaries and zero-crossings regardless of translation, rotation and scaling [Boles93]. Boles and Tieng plan to continue this research on 3-D objects.

1.5 SOFTWARE AND HARDWARE FOR THE WAVELET TRANSFORM

Why not use software to perform the wavelet transform? Software, such as Tsunami Plus, transforms data to a wavelet representation. Tsunami Plus is a library of C routines developed for the non-specialist to use. Wavelets can be used to represent data as diverse as heartbeats and fingerprints [Smith94]. Matlab [Matlab96] has developed a wavelet package to allow users to easily perform wavelet transforms on 1 or 2 dimensional matrices. Software has the advantages of flexibility and ease of use, which make for a good learning environment for the scientist. However, dedicated hardware will always outperform software when it comes to speed. Using commercially available image processing hardware, an engineer can build a wavelet transform system, though the best results will be from chips designed just for the DWT. Since most applications rely heavily on speed, this paper will focus on hardware, especially architectures specific to the DWT.

The 1-D and 2-D wavelet transforms, both DWT and Continuous Wavelet Transform (CWT), have been implemented with single chip architectures. Signal analysis applications use the CWT, while compression and numerical analysis applications use the DWT. The continuous wavelet transform (CWT) varies from the DWT by not downsampling the filter outputs, and the CWT architectures are fairly similar to the DWT architectures. As such, they will not be discussed in this paper. The 1-D Inverse Discrete Wavelet Transform (IDWT) has also been fabricated on one chip. Due to the only recent interest of VLSI professionals, no optimal chips are yet available for programmable wavelets [Chakrabarti96]. Fridman and Manolakos, however, have developed an optimal 1-D DWT architecture [Fridman97].

The Houston Advanced Research Center (HARC) developed an image and video codec

based on Boundary Spline Wavelets. Called HARC-C, the codec can compress data in a variety of formats, such as audio, black-and-white images, 24-bit color images, and video. It compares favorably to the Joint Picture Experts Group (JPEG) and Motion Picture Experts Group (MPEG) standards. It outperforms JPEG by as much as 3 times the compression ratio. The HARC-C and MPEG are compared, using software only, at a video rate of 30 frames per second (fps). The MPEG needs at least a 133 MHz Pentium machine to play video at this rate. Meanwhile, the HARC-C is able to play a video almost 4 times the size on a 486 machine running at 66 MHz. To compress data, the HARC-C needs 1.4 times the amount of time that it takes to decompress the data. The University of Texas Anderson Cancer Center is looking at using HARC-C to store and transmit X-ray images. Another expected use for the HARC-C is CD-ROM publishing [Ozer95].

VDOnet Corporation released a video product called VDOLive for PCs running the Windows operating system. Macintosh and Unix versions should be released in 1996. VDOLive uses wavelets to compress video. A server transfers the compressed video over a communications link, such as the Internet, to a client running the VDO Player. The server software adjusts the amount of data transferred to the user by monitoring the current bandwidth, and sends the video frames in layers of detail. A slow communication link results in a degraded video frame, but the audio and motion is more continuous. Tests indicate that the communication link would need to be at least a single ISDN line (running at 64 Kbps) to achieve playback of 15 fps [Ozer96].

Since the DWT basically has multipliers, adders, and memory, why not simply make a design with these three components? Szu, et al. designed a straightforward filter with multipliers, adders and flip-flops [Szu94]. Using these filters, one can make a 1 or 2-D DWT. The design achieves downsampling and upsampling with flip-flops. This design was made for small data values, where the input data and the wavelet coefficients are both 4 bits wide. A data value can only range

from -7 to +7. Other applications would need wider busses for the data and wavelet coefficients.

The discussion of the architectures centers on the design of analysis hardware. The synthesis hardware can be built similarly. Each architecture has advantages and disadvantages, so all are valid alternatives depending on the application. Using these architectures, a designer can choose the features and number of octaves, which best fit, the desired application.

1.6 THE BASIC 1-D DWT

The DWT multiplies the input by the shifts (translation in time) and scales (dilations or contractions) of the wavelet. Below are variables commonly used in wavelet architecture literature.

J is the total number of octaves

j is the current octave (used as an index. 1 <= j <= J)

N is the total number of inputs

n is the current input (used as an index. 1 <= n <= N)

L is the width of the filter (number of taps)

k is the current wavelet coefficient

W(n,j) represents the DWT, except:

W(n,0) which is the input signal.

The low-pass output is:

$$W(n,j) = \sum_{m=0}^{2n} W(m,j-1)*lowfilter(2n-m)$$

and the high-pass output is:

$$W_h(n,j) = \sum_{m=0}^{2n} W(m,j-1)*highfilter(2n-m)$$

The fast pyramid algorithm, by Stephanie Mallat and Yves Meyer [Mallat89], quickly computes the 1-D wavelet transform. The algorithm gets its efficiency by halving the output data at every stage, otherwise known as down sampling. The algorithm has a complexity of O(N), with an input of N samples. Other transforms typically need $O(N^2)$ calculations. Even the Fast Fourier Transform takes O(N*logN) computations. The number of wavelet coefficients plays a large role in the time taken to compute the wavelet transform. Since every input sample must be multiplied by each wavelet coefficient, the number of coefficients should be kept to a minimum. On the other hand, more coefficients allow for a better approximation. The inverse wavelet transform reconstructs the signal from wavelets, which perfectly matches the original. The reconstruction lasts approximately the same amount of time as the wavelet analysis.

Note that every octave divides the value n by 2, since the DWT outputs are downsampled at every octave. In other words, the input length, N, equals 2^J, and the DWT generates $N/2^j$ outputs for each octave. However, practical applications limit the number of octaves. The number of operations can be derived from this information and is proportional to 2t(1-(1/N)), where t represents the number of operations in the first octave. Therefore, the DWT has a lower bound of Omega(N) multiplications [Vishwanath92].

Knowles presents a simple architecture for the 1-D DWT [Knowles90]. His contribution is significant because it is the first. He shows the DWT as a case of subband coding. With scheduling, only two filters, a low pass and a high pass, are needed to perform the DWT. Based on this

observation, Knowles draws up a schedule and designs a fully pipelined circuit to implement it. Note that since a DWT filter only keeps half of the filter outputs, due to downsampling, then only half need to be computed [Knowles90]. The inputs feed to the filters at a constant rate, but outputs exit the filters at half the rate. The time between outputs can be used to compute outputs for other octaves. With this in mind, a pipelined DWT processor can be designed around a pair of filters.

Figure 1.1 shows a DWT for 4 coefficients, which is equivalent to subband coding. The four wavelet coefficients are labeled a, b, c, and d. The input signal goes to the low-pass and high-pass filters of the analysis stage The high-pass filter multiplies the input signal by a transposed vector of the wavelet function coefficients. The low-pass filter multiplies the input signal by a transposed vector of the scaling function coefficients. The outputs of each filter are downsampled, meaning that every other value is thrown out. At this point, the signals can be compressed, or transmitted, or processed as desired. For the sake of example, figure 1.1 shows the signals feeding into the synthesis stage, to reconstruct the original data. This is also known as a Quadrature Mirror Filter (QMF). In the synthesis stage, the signals are first upsampled. Upsampling stretches the data out, inserting a value of 0 between each signal value. Next, the signals are multiplied again by vector transposes in parallel, and the results are added together. The output signal should be the same as the input signal, except for a constant factor. An unvarying input signal demonstrates this point.

Suppose the input signal consists of N samples, and each sample has the value 1. The output signal will be 0, c0, c0, c1, c1, c2, c2, ... c2, c1, c1, c0, c0, 0, where c0 = ac + bd, c1 = aa + bb + cc + dd + c0, and c2 = c0 + c1. One requirement of a wavelet is that ac + bd = 0, so that the output signal will become 0, 0, 0, c3, c3, ... c3, c3, 0, 0, 0, where c3 = aa + bb + cc + dd. There will be N values of c3. Typically, as in the case for the Haar wavelet, the value of c3 is 2. Thus, every output value needs to be divided by the c3 value, called a scaling factor, to get the correct reconstruction. The

input signals have the leading and trailing values wrapped around, in order to keep the output signal the same size as the input signal. In an application such as a VLSI chip, the leading and trailing input signal values may not be available at the same time, but this should not adversely affect the output signal.

Multiple levels of resolution (called octaves) are possible [Mallat89]. The low-pass outputs feed to another set of filters. Likewise, the low-low outputs feed to another filter set. This process can be repeated until only one output remains from the lowest octave. However, typical applications do not need this level of analysis. Figure 1.2 shows 3 octaves, which is a good number for demonstration purposes. The actual number of octaves depend on the application, with as many as $\log_2 N$ possible octaves.

1.7 THE 2-D DISCRETE WAVELET TRANSFORM

Why use a 2-D DWT? Why not just treat it as a 1-D signal? The 2-D DWT gives better compression. Also, 1-D representation of 2-D data introduces edge effects, or artificial discontinuities. An application with 2-D data need a true 2-D DWT. 1-D is used in speech and music, while 2-D is used in image compression. The 2-D differs from the 1-D by going in both the X and Y directions, shown in Figure 1.3 for 1 octave. Even 3-D data, such as video, can use a 2-D DWT followed by difference encoding. However, the 3-D DWT outperforms the 2-D DWT [Wang95].

"For a m-level two-dimensional wavelet, the average number of filtering operations is given by $2 + 2/4 + 2/16 + ... + 2/4^{(m-1)} = (8/3)(1 - 4^{-m})$. In other words, the upper bound on the number of low-pass and high-pass filters required for the two-dimensional wavelet operation in 8/3. Since the

number of low-pass and high-pass operations are the same, the upper bound on the number of low-pass or high-pass filters is 4/3. In a complete word-level realization, we would require 2 low-pass and two high-pass filters" [Parhi93].

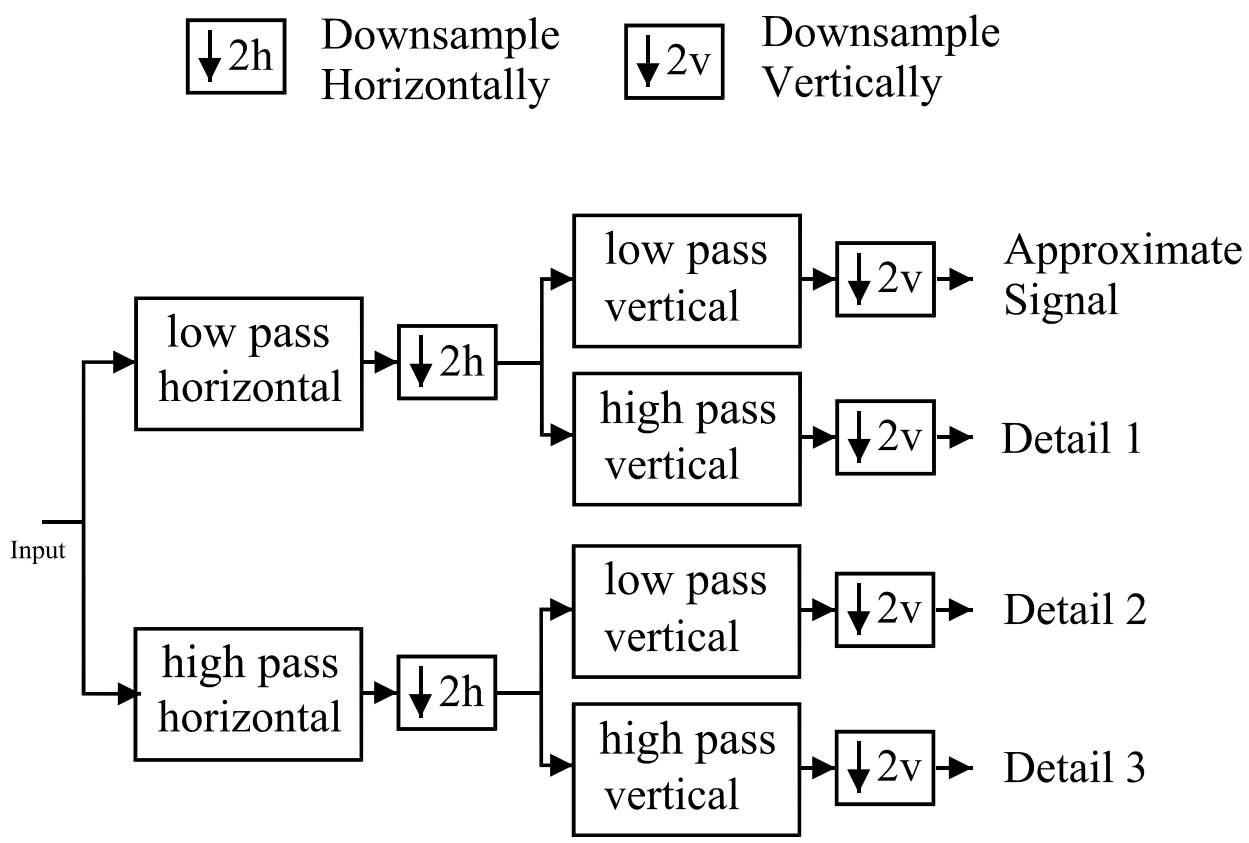


Figure 1.3 - A 2-Dimensional, 1-octave DWT

Edge effects appear due to the boundaries of the input. Since the input has N samples, the output from a Finite Impulse Response (FIR) filter has N + L - 1 samples. How the DWT algorithm should behave at the boundaries is unclear, but there are several solutions available. Fridman and Manolakos say that the solution depends on the application, and is still an open problem [Fridman94b].

Input data flow concerns the designer of a 2-D system, since the input stream should be continuous. The hardware scans the input image from left to right horizontally, but the edges present a problem. One possible solution scans the image in a raster fashion, connecting the start of the next line with the end of the previous line. However, the edge of the image introduces a discontinuous

transition. Instead, Lewis and Knowles [Lewis91] recommend snaking through the image. When the left to right scan completes, a right to left scan begins on the next line. The problem with this solution is that it requires the wavelet coefficients to be reversed as well, since the filters are non-linear phase. Lewis and Knowles show that reversing the signs of the wavelet coefficients, followed by doubling the output values in the reconstruction, generates the correct values. The vertical direction presents another challenge since the data are separated into low and high bands, which should be scanned independently of each other. This only introduces one discontinuous point [Lewis91]. In this work we assume a scan in raster fashion, as do most other architectures. In addition, the wavelet transform handles discontinuities well.

The Discrete Wavelet Transform (DWT) comes from Mallat's multiresolution pyramid algorithm, explained above. It can be performed on 1, 2, or 3-dimensional data. This chapter explains the DWT algorithm for the different types of data. Also, this chapter discusses compression based on the DWT.

Figure 1.3 shows how the DWT operates on a 2-dimensional image. First, the input signal feeds to both high-pass and low-pass filters. The first filters operate horizontally, meaning it treats the image as Y vectors of length X, where X and Y are the length and width of the image, respectively. After the initial filters, the horizontal downsampling procedure discards every other column. The downsampled outputs feed into the second set of filters. These filters treat the incoming data as X/2 transposed vectors of height Y. Next, the vertical downsampling procedure discards every other row. The results of the 2-dimensional, one-level DWT are 3 detail images and 1 signal approximation. Each resulting image is one-fourth the size of the original image. Also, calling the results "images" is erroneous since they contain invalid greyscale values, such as negatives. A greyscale value has the range 0 to 255. However, these resulting values can be mapped to valid

greyscale values and viewed, if desired. A multi-level DWT can be implemented by treating the signal approximation as a new input signal. The outputs from the next DWT level will be one-fourth the size of the original. The synthesis of the results appears as a mirror image of figure 1.2, where the downsampling is replaced by upsampling. The first upsampling unit inserts rows of 0's between its input rows, and the second one inserts columns of 0's. After every recombination, the synthesizer divides the compensation factor (c3) from the resulting signal.

1.8 WAVELET PACKETS

Wavelet packets and cosine packets, also called time-frequency waveforms, are alternate transforms that fall in between the wavelet and Fourier transforms. The packets are like wavelets since they have limited duration, and are shifted in time to the spot needed, but they are also like sinusoids in that they have frequency. Packets better represent signals that oscillate locally than the other transforms. In an application of wavelet and cosine packets, much of the noise was removed from an 1889 recording of Johannes Brahms playing the Hungarian Dance Number 1 in G minor [Bruce96]. Researchers also use wavelet and cosine packets to compress fingerprint information to a ratio of 20:1, four times better than the JPEG compression ratio.

Walczak, et al. discusses the Wavelet Packet Transform (WPT) as a tool for Infra-Red (IR) pattern recognition [Walczak96]. The Wavelet Packet Transform can be considered a more general case of the Discrete Wavelet Transform. The two transforms are alike in that they both recursively analyze the approximation signals from the low pass filters. The WPT recursively analyzes the detail signals as well. Then, the WPT keeps a complete representation of the signal based on the best concentration of energy. The DWT differs by always keeping the same representation pattern

[Walczak96].

The WPT decomposes a signal into a tree of transforms. The original signal breaks down into a low frequency component (the approximation) and a high-frequency component (the detail). Each is half the length of the original. These two components form the first octave. Then the low-frequency component goes through the same filter pair, forming a low-low component and a low-high component. Likewise, the high-frequency component is filtered, forming the high-low and high-high components. These 4 components form the second octave. Note that each component of the second octave is 1/4 the size of the original. In other words, the length of the four components put together equals the length of the original signal. This process continues for as many octaves as desired, or until the components are only one value in length. The decompositions form a tree, where one level of the tree represents one octave. The tree contains many potential ways of representing the signal. Each potential representation must cover the whole width of the tree, but not overlap in height. In other words, once a component has been chosen to be included in the representation, then no parent component of it can be included. Also, no child component of it can be included for the same reason: the representation would be redundant. The DWT keeps the high component from octave 1, the low-high component from octave 2, the low-low-high component from octave 3, and the low-low...low-high and the low-low...low-low components from the Jth octave. Note that the DWT only keeps one low (approximation) component. The DWT does not analyze the high pass filter components. Thus the DWT is just a special case of the WPT [Walczak96].

How does one choose which components from the WPT tree to keep? Given a potential solution, the energy and entropy quantify how good the solution compares to others. The energy of a signal is the sum of the squares of the signal's coordinates. The entropy measures the distribution of the energy. Low entropy means that a few components contain most of the energy, and low entropy

makes for good compression. By comparing the entropy of a component to its 2 child components, Coifman's best-basis algorithm [Coifman93] allows one to choose the lowest entropy solution [Walczak96].

Typically, the DWT concentrates most of the signal's energy into the low pass output. Thus, even if one were to use the WPT, the output would most likely be the same as the DWT. Under this assumption, the DWT is best since it accomplishes the same result with fewer calculations. The WPT is worth mentioning since the WPT is similar to the 3-D DWT except that the 3-D DWT takes all three dimensions into account. In other words, a 3-D DWT processor could be rewired to form a 1-D WPT processor (without the evaluation hardware to select which representation is best).

1.9 THE 3-D DISCRETE WAVELET TRANSFORM

The 3-D DWT must be separable, which means that the 3-D transform is accomplished by a 1-D DWT in each dimension. The scaling and wavelet functions, Phi(x) and Psi(x), are given below for the 3-D case [Wang95]:

Phi(x,y,z) = Phi(x)Phi(y)Phi(z) (scaling)
Psi1(x,y,z) = Phi(x)Phi(y)Psi(z) (wavelet 1)
Psi2(x,y,z) = Phi(x)Psi(y)Phi(z) (wavelet 2)
Psi3(x,y,z) = Psi(x)Phi(y)Phi(z) (wavelet 3)
Psi4(x,y,z) = Phi(x)Psi(y)Psi(z) (wavelet 4)
Psi5(x,y,z) = Psi(x)Phi(y)Psi(z) (wavelet 5)
Psi6(x,y,z) = Psi(x)Psi(y)Phi(z) (wavelet 6)
Psi7(x,y,z) = Psi(x)Psi(y)Psi(z) (wavelet 7)

The 3-D WT is like a 1-D WT in three directions. Refer to figure 1.4. First, the process transforms the data in the x-direction. Next, the low and high pass outputs both feed to other filter

pairs, which transforms the data in the y-direction. These four output streams go to four more filter pairs, performing the final transform in the z-direction. The process results in 8 data streams. The

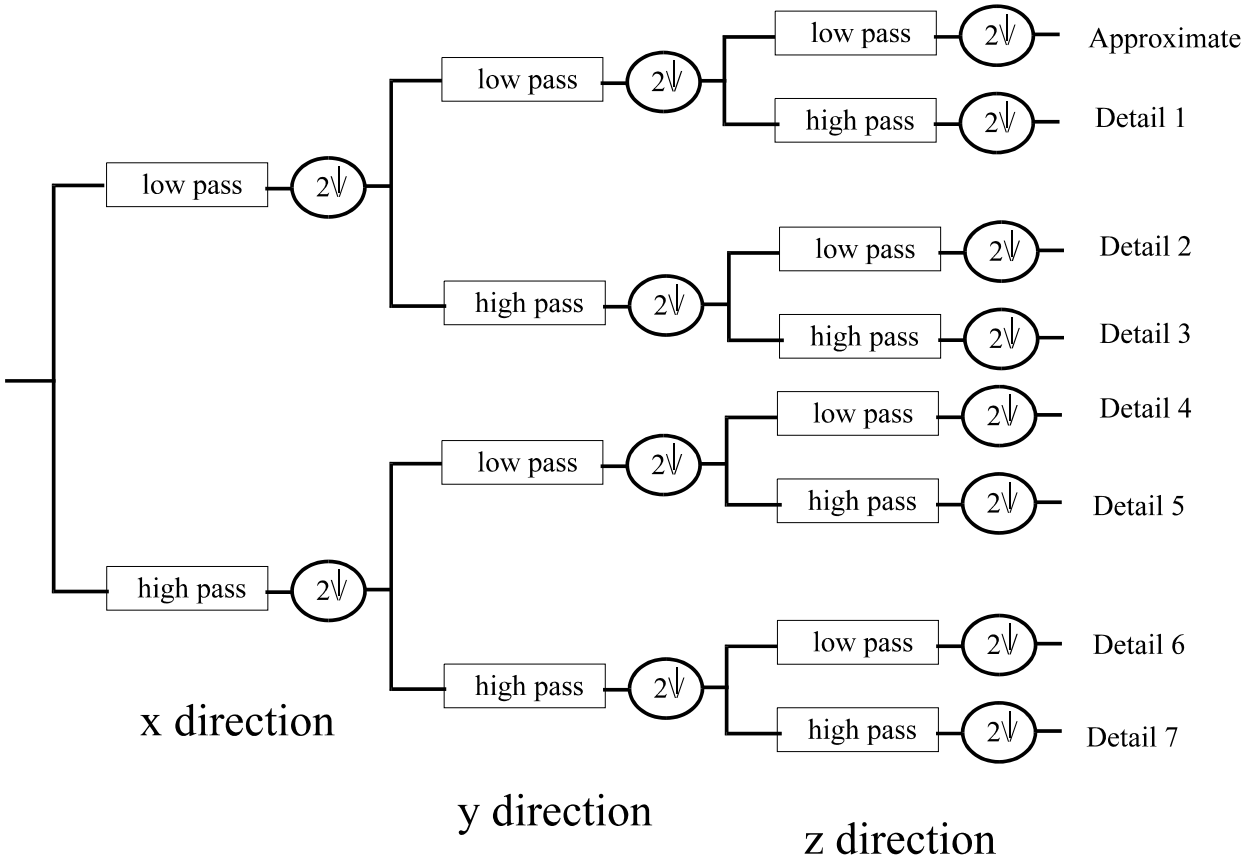


Figure 1.4 - The 3-D DWT

approximate signal, resulting from scaling operations only, goes to the next octave of the 3-D transform. It has roughly 90% of the total energy [Wang95]. Meanwhile, the 7 other streams contain the detail signals. Note that the conceptual drawing of the 3-D WT for one octave has 7 filter pairs, though this does not mean that the process needs 7 physical pairs. For example, a folded architecture maps multiple filters onto one filter pair.

Three-dimensional data, such as a video sequence, can be compressed with a 3-D DWT, like

the one shown in figure 1.4. Most applications today perform a 2-D DWT on the individual images, then encode the differences between images (called difference coding). Given the first image, followed by the differences between the first and second frame, the second image can be constructed. For an image sequence with little change, such as a video of a person speaking, this method works well. For other sequences, such as medical imaging, the image sequence will have more changes between frames. Thus a true 3-D method would work better. Table 1.2 below shows the typical image sizes (first two dimensions) for medical imaging applications. While the 3-D architectures presented here are aimed at the MRI application, they could possibly be used with other medical applications, or video of any kind.

Why do we need a 3-D DWT? DWT considers correlation of images, which translates to better compression. According to Lee, et al., the 3-D DCT is more efficient than the 2-D DCT for x-ray CT [Lee93]. Likewise, one would expect the 3-D DWT to outperform the 2-D DWT for MRI. Wang and Huang showed the 3-D DWT to outperform the 2-D DWT by 40-90% [Wang95]. The DWT has advantages over the DCT. First, DCT difference coding is computationally expensive. Second, wavelets do not cause blocking artifacts, which are unacceptable in medical images. The DCT is not optimal for medical image compression, even if it performs well on video [Villasenor93]. Currently, 3-D transforms have been done with other methods, like a network of computers, but a chip dedicated to this transform will give faster results. Due to better medical input devices (scanners) interslice distances get smaller, improving the correlation between images, resulting in even better results for the 3-D transform.

Medical Application	Typical Image Size

Magnetic Resonance Imaging (MRI)	256 x 256
Positron Emission Tomography (PET)	128 x 128
Computerized Tomography (CT)	512 x 512
Digital Subtraction Angiography (DSA)	512 x 512

Table 1.2 - Typical image sizes of medical applications

Why use Wavelet Transforms with Medical applications? Medical applications deal with vast amounts of data as the following example illustrates. Lee, et al. reports that "the University of Washington Medical Center, a medium-sized hospital with about 400 beds, performs approximately 80,000 studies per year. At 30 Mbytes per study, the amount of digital images generated is 2.4 Tera (10^{12}) bytes of data per year or approximately 10 Gbytes per day" [Lee93]. The wavelet transform allows for a good deal of compression without losing diagnostic accuracy. As Angelidis notes, MRIs "must be stored for long periods of time both for medical and legal reasons" [Angelidis94].

Picture Archiving and Communications Systems (PACS) are replacing film archives of medical images. PACS network the input devices, output devices, and storage facilities (i.e. databases) [Lee93]. PACS store the data digitally. Storage and transmission systems need compression for more efficient operation. Finally, wavelet encoded data naturally allows progressive transmission, such as sending the "average" data first followed by the details. The end user has a rough idea of what the final output will look like, and can request more details as needed [Riskin90].

1.10 THE FOCUS OF THIS WORK

The main focus of this dissertation is to develop dedicated architectures for the 3-D DWT. Compression is explained more thoroughly in the next chapter. The third chapter surveys existing (1-D and 2-D) DWT architectures. Following the existing DWT architectures, the fourth chapter

presents new architectures for the 3-D DWT. The analysis and results follow in chapter 5, and chapter 6 presents the conclusions.

2. COMPRESSION

2.1 COMPRESSION INTRODUCTION

If the computer stores information differently, can the amount of space needed to represent the data be reduced? The answer is yes. There are several algorithms to store information differently, such as Huffman coding. If the information can be approximated with fewer bits, then the bits can be strung together to take up less space. For a large signal, even a small change can make a big difference. For example, consider an image stored in a straightforward manner, where one byte represents the greyscale value. A file of 256x256x8 bits, or 65,536 bytes would be needed. Now suppose that the storing program maps the pixels to 128 values instead of 256. The file would be 256x256x7 bits, or 57,344 bytes large, saving 12.5% of the space required. There would need to be some header information to store the mapping, but this is negligible for a large file. For example, an additional 256 bytes of header data should be sufficient to provide the mapping above, which still allows a savings of 12.1%. If the original image has more than 128 pixel values, then this compression loses information and is called "lossy". The signal should be compressed only when the uncompressed signal has a tolerable amount of error when compared to the original. The tolerable amount of error for the purpose of medical diagnosis is measured by human perception, though more rigorous methods are available. A threshold function can eliminate transformed values that occur infrequently, or are very small. This gives a good amount of compression (space savings), while giving perceptually good results. The thresholding process is called quantization.

Imagine a signal with only two values. A binary signal can represent it, requiring only 1 bit per value. For example, a black and white version of a 256x256 image would have values of 0 and 255 only, yet would be stored as 65,536 bytes. A 256-byte header, followed by a bit stream of 0's

and 1's would be able to store the same image using only 8448 bytes, a savings of 87%, or a compression ratio of 7.8:1. Note that this case is loss-less, meaning that the original and reconstructed images are exactly the same. Typical compression loses information in the quantization step.

The compression operation takes place after the wavelet analysis. Before the synthesis, the signal is uncompressed. Compression has 3 steps: transformation, quantization, and entropy coding. The transform considered here is the DWT, although other transforms such as the Discrete Cosine Transform could be used. Why not do compression without the transform? The transform de-correlates the original signal so that it will give a better compression ratio. The quantization and entropy coding are discussed below. Transcoding is also discussed, since some applications require it.

2.2 QUANTIZATION

Quantization is the second step used in compression. It works by reducing the precision of the data, which naturally means that some information will be lost. Note that the first J-1 low-pass outputs are discarded, since the synthesis filters reconstruct these signals. The low-pass-J output and the J high-pass outputs are all the information needed to build the output signal. The quantization process leaves the approximate signal alone since it contains so much of the total energy and works with the detail signals.

There are two types of quantization: scalar and vector. While vector quantization may lead to a better compression ratio, it is more complex than scalar quantization. For this study, scalar quantization is used unless otherwise specified.

A quantization algorithm has several steps. First, the algorithm zeroes-out all data values whose absolute value is less than the chosen threshold value, T_m, where m represents the octave number. A quantization number, Q_m, affects the mapping by dividing the difference of the data value and T_m. The next step of quantization rounds the detail signal's floating-point values to integer values. The rules below describe the quantization mapping process, where "data" means the data value after the wavelet transform, and "q-data" means the new, quantized value for this data point. These data can be multi-dimensional, but the subscripts were left off for simplicity [Wang95].

$$\text{q-data} = \text{round}((\text{data} - T_m)/Q_m) \qquad \text{if data} > T_m$$

$$\text{q-data} = 0 \qquad \text{if } -T_m <= \text{data} <= T_m$$

$$\text{q-data} = \text{round}((\text{data} + T_m)/Q_m) \qquad \text{if data} < -T_m$$

Vishwanath and Chakrabarti consider two on-line coding schemes [Vishwanath94a]. The first scheme uses scalar quantization, while the second one uses vector quantizing. The second scheme needs a raster-to-block converter since the output of the 2-D transform is in raster form, and the vector quantizer (VQ) uses blocks of data. In a typical case with one-dimensional blocks, the codebook used with the VQ is much larger than the storage needed for the raster-to-block converter. In a typical case with a 2-dimensional block size, the design requires VQ-codebook-size + RAM-needed-by-the-transform + RAM-needed-by-raster-to-block-converter bytes of storage. The storage requirements can be expressed with the following parameters: the input frame size is N x N, the precision is k, and b is the block size. The transformer needs 2LNk bytes of RAM, while the raster to block converter needs 3bNk bytes. Typical values for the parameters are N=640, L=4, k=16, and b=4. Therefore, this design needs at most 24 Kbytes of storage in addition to the VQ codebook.

2.3 ENTROPY CODING

The final compression step is entropy coding, which involves run-length coding and Huffman encoding. Run-length coding checks for a series of repeated values, then represents them with two numbers, series length and series value. For example, imagine a 1-D data sequence of 0, 0, 0, 0, 0, 128, 128, 128. In run-length encoding, this sequence of five repeated 0's and three repeated 128's becomes simply 5, 0, 3, 128. One variation is to code the differentials of a stream, which generates more zeroes in the resulting encoding. The numbers indicate the amount to add to the accumulated value, for example, the data sequence above would become 0 + 0, 0, 128, 0, 0. This variation has the advantage of producing more runs of zeroes. The run-length encoding process should reduce the amount of data since the quantization process produces a lot of 0 values.

The next step uses Huffman coding to further compact the resulting data. Huffman coding analyzes the data and notes the frequency of the data values. Based on this frequency, it allocates bits to the representation. A typical greyscale image, for example, has 256 possible values, and uses 8 bits to represent each value. Huffman coding uses only a few bits to represent common values, and more bits to represent uncommon values. A natural example of this idea can be found in the TV game show "Wheel of Fortune", where the contestants always guess letters like "c" and "s" first since they are very common in English words, but rarely guess "x" or "z". Consider, for example, if the alphabet were represented with 8 bits for each letter, and we were to Huffman encode a text document. Huffman coding would assign less than 8 bits for "c" and "s", but possibly more than 8 bits for "x" and "z". The ratio of the total number of bits used to the number of letters in the document would be less than 8. Of course, which letters are assigned to which codes depend on the

document. In this way, Huffman coding reduces the average number of bits-per-data value, indicating a savings in space. In the above greyscale case, reducing the average number of bits-per-data to 4 cuts the overall amount of storage needed in half. Of course, the Huffman encoded data needs a small look-up table to indicate which data values are most frequent.

2.4 TRANSCODING

A transcoder may be needed as well, if the application requires it. For example, multi-casting video for a teleconference over multiple-rate channels requires a transcoder to deal with the differing line speeds. Since the application is interactive, the transcoder should work on-line [Vishwanath94a]. The video encoder cannot help the network latency, so designers must concentrate on minimizing the latency in the Encode-Transcode-Decode (ETD) process. Other design considerations are: making the software and hardware economical, making the transcoding level variable, and making the transform coding compatible with a variety of possible coders.

The encoder takes an input frame and performs the DWT. It outputs octaves 1 through J to the network. The transcoder interfaces the encoder with the decoder. The decoder receives as many of the octaves as possible given the data rate, with the lower octaves (approximate signal and broader details) taking precedence over the higher octaves (finer details). Thus, the connection speed directly affects video quality [Vishwanath94a].

2.5 COMPRESSION PERFORMANCE

Two formulas evaluate how well the compression works. The Peak Signal to Noise Ratio

(PSNR) determines the quality of the images. The Compression Ratio (CR) shows how much the data reduced during compression. The size of the original image divided by the size of the compressed image gives the CR.

$$CR = \text{size of original image} / \text{size of compressed image}$$

The PSNR is a little more complex. First define the Root Mean Square Error (RMSE), which involves comparing the original image pixels to the uncompressed image pixels [Wang95]. The RMSE =

$$\frac{\text{sqrt}(\Sigma(\text{original\_image}(x,y,z) - \text{uncompressed\_image}(x,y,z))^2)}{\text{total number of pixels}}$$

The equation below gives the PSNR.

$$PSNR = 20 \log (\text{Maximum grey level in image} / RMSE)$$

While these equations give a comparison between compression amounts, they are not the only criteria used to judge whether a certain compression is good. Ultimately, a human observer will be able to tell a good compression from a bad one. In the case of medical images, a doctor would need to judge whether an image is still good for diagnosis. As an extreme example, consider the corners of the MRIs. No medical information is contained here, but if an entire corner's information were lost in the compression procedure, the resulting image would have a poor PSNR, though the rest of the image would be fine.

2.6 2-D COMPRESSION

The statistical ways to compress an image rely on removing redundancy from the image. This suffers from two problems. First, these methods can only remove so much redundancy before removing information that is non-redundant. Second, the reconstructed image tends to have degradation that can be seen with the eye. A better compression method exists, which imitates the human visual system (HVS). The human visual system relies on edge detection as the best way to recognize an image. Experimental results on the HVS show that the eye decomposes images into frequency bands and scales in space. Thus, the orthogonal wavelet transform, in the form of a quadrature mirror filter, operates on the image to imitate the HVS. Marr said that, "the physical phenomena that give rise to intensity changes in the image are spatially localized" [Marr82]. Based on this observation, Lewis and Knowles present a compression algorithm [Lewis92].

Lewis and Knowles use a 2-D, 4 coefficient, Daubechies wavelet transform. The coefficients are 11/32, 19/32, 5/32 and -3/32. The transform has four octaves, which generate 13 subbands. Note that it needs the same amount of memory for the 13 subbands combined as it does for the original image. In other words, the storage requirements have not changed. However, the next step compresses the subbands, except for the low-pass subband. Marr's locality observation comes into play here [Marr82]. The program separates the important edge data from noise by looking at multiple octaves. A re-occurring detail is important. The algorithm looks at the lowest resolution images for a guess as to where the edges are, then checks the higher resolution details to confirm the guess. To guess where an important detail lies, the algorithm compares the image to a threshold value, which was obtained through an HVS model.

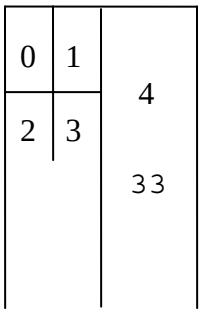
Lewis and Knowles note that each datum in a high-pass band in the lower octaves has 4 data points in the corresponding locations in the octave above. This makes sense, since the subbands on a

lower octave are one-fourth the size of the image above them. Thus, Lewis and Knowles use a 4-branch tree structure. The four branches above a point are only considered if the absolute value stored at the point's location exceeds the threshold. They concede that it would be a better measure to look at the energy level around the point to avoid thresholding errors.

The HVS is modeled mathematically, taking into account the noise factors that would affect the human eye. These factors include background luminance, edge proximity, band sensitivity, and texture. Applying this model, the Lewis and Knowles algorithm quantizes the high-pass subband data. At this point, Huffman coding can compress the quantized data.

Lewis and Knowles assert that their method achieves better compression ratios than standard subband compression techniques. They claim it can be implemented simply in hardware. Since the HVS model adds noise in the image only where it is least noticeable, the compression and resulting image quality are improved over the other compression methods [Lewis92].

Adaptive image compression allows for variable bit rate encoding. The idea is to prioritize the data and send the most important information first, followed by the lesser important data if bandwidth allows. Acharya, et al. presents an architecture for this application. First, it transforms the digital 2-Dimensional image into two levels of low-pass and high-pass components. Figure 2.1 shows the DWT for two octaves, which results in 7 subband signals. Next, the components are encoded into a compressed form based on bit rates.



5 6

Figure 2.1 - The 2-D decomposition of an image into 2 octaves

More bits are assigned to the more significant parts of the image. An encoding scheme such as Huffman coding operates on the final blocks.

The next stage classifies the DWT outputs. Classification labels similar blocks with the same class. For example, Acharya, et al. chose spectral classification for their architecture since it is less computationally expensive than gain-based methods [Acharya97]. The gain-based methods require sorting, which adds to its complexity, while the spectral classifying methods do not. In spectral classification, the residual energy and autocorrelation values are used to decide whether a block should be in class 0 or class 1. The goal is to find the best auto-regressive (AR) model for each block out of a small collection of possible models. Acharya assumes a block size of 4x4. Of the 7 subbands, the lowest frequency subband (LFS) does not get classified. The LFS is the most similar to the original image. The other subbands are called high frequency subbands (HFS), since the results have gone through at least one high pass filter. Each HFS is split into two data sequences, one for each class. Thus, there are 13 data sequences total. The more important sequences are assigned higher bit rates.

The bit rates are assigned with minimal total distortion in mind. The Steepest-Descent algorithm solves the bit rate allocation problem. The variances are computed by circuitry, and these variances are multiplied by amounts from the rate-distortion table. Thirteen registers store the results. Next, a module finds the maximum of the 13 register values. The rate corresponding to the

maximum register increases to the next available rate. The new rate produces an updated value in the rate-distortion table, which results in a new register value. This process continues until the total bit rates have reached the design rate. The next available rate is found in practice by setting the increasing value to a constant 0.2 bit. The final bit rates are used to take advantage of variable length coding [Acharya97].

2.7 COMPRESSION OF 3-D DATA

Wang and Huang [Wang95] demonstrate the need of medical applications for 3-D compression for transmitting large amounts of video data quickly. The DWT compression will give the best (most visually pleasing) results while still allowing a good compression ratio. It is assumed that “most visually pleasing” and “diagnostically accurate” are the same. Lewis and Knowles recommend a Haar wavelet for the time dimension, while the spatial dimension's wavelet will be up to experiment. Szu, et al. recommend a 7 and 9 coefficient biorthogonal wavelet, while Lewis and Knowles find pleasing results with the 4 tap Daubechies wavelet.

Medical resonance images (MRI) are a series of 2-D slices used in medical diagnostics. Compressing this volume of data is important since it needs a large amount of storage space and/or bandwidth. A 3-D wavelet transform takes advantage of the fact that the third dimension is related to the first two dimensions to provide a more efficient transform [Pratt96]. Each MRI pixel in the test sequence is a 1 mm x 1 mm x 4 mm section of a human head, with 4 mm being the gap size between slices. Thus, the MRI completely maps out a 3-D body part as a 3-D block of data. The test sequence has 53 slices of 256x256 pixels, and each pixel value has a width of 8 bits. The output from the 3-D wavelet transform chip can be quantized then encoded for a good amount of compression, for

example, Pratt, et al. uses zero-tree encoding [Pratt96]. The 3-D DWT chip could be used for other applications as well. Though MRI data specifically benefit from this transformer, the chip would be able to work on any 3-D signal, such as a greyscale (or monochrome) video clip.

The number of images that form a set vary, but the number of images do not matter to the architectures, except to allow them to know when the transform is done. As the previous chapter showed, the 2-D architectures are dependent upon the image size. The dependence comes from the fact that they save intermediate computations, equaling the size of the width of the image.

Medical applications were chosen as the target application for the 3-D Discrete Wavelet Transformer. Medical applications desperately need a powerful transform to compress the volumes of data. Luckily, MRIs are of reasonable X and Y dimensions, meaning that one MRI could be stored in a RAM. While the 3-D DWT would work well in television applications, one chip may not be able to store the greater image size. Also, MRI data has greyscale pixel values, so the data precision (8 bits) is less than the bits per pixel of a TV signal. Since the TV signal has color, one chip may not suffice (though 3 of them in parallel for red, green and blue might). Also, MRI data does not have an accompanying sound track, so there is no need to compress a 1-D sound signal as well. A sound signal may need synchronization with TV images after the decompression. Therefore, while television applications may be on the horizon, medical applications are the current target for this technology.

Wang and Huang apply the three-dimensional wavelet transform to medical imaging as part of a compression algorithm. To speed up the process, they implement it on a network of workstations. Comparing the results to 2-D WT compression, the authors find that the 3-D transform performs 40-90% better [Wang95].

Currently, researchers compress 3-D data with 2-D DCT difference coding. In other words,

the computer subtracts one 2-D image from the next image in the sequence, and encodes the difference with the DCT. This compression does not take into account the correlation between images. It also suffers from the blocking effects that haunt DCT coding. Finally, the process requires many computations. By using the 3-D DWT, one can achieve a higher rate of compression without the blocking effects.

Wang and Huang [Wang95] numerically compare the performance of the 3-D DWT compression method to the 2-D DWT method. Using the formulas given above, the compression ratios of the 3-D method outscore the 2-D method by 40-90%, depending on the PSNR value. The 3-D method works much better especially at lower PSNR values. For best compression, the 3-D DWT compression method outperforms the 2-D DWT method.

Next, Wang and Huang used a local area network of workstations to calculate the 3-D compression, since it requires so many computations [Wang95]. This approach greatly speeds up the process. A master computer parcels out the computations and assembles the results. The slave computers do the work of compression, operating in parallel. The authors compare the speedup of the 3-D method achieved by making the computation parallel. Let T_1 be the time needed by one processor to do the compression, and let T_n be the time needed by n processors.

$$\text{Speedup} = T_1 / T_n$$

$$\text{Efficiency} = \text{Speedup} / n$$

The speedup and efficiency equations measure the success of making the compression parallel. The authors found a roughly linear speedup as the number of processors increase. The efficiency, which

measured between 80% and 90%, drops as more processors are added, since network traffic becomes more of a bottleneck [Wang95]. Thus, the parallel implementation efficiently decreases the time needed to compute the 3-D WT compression.

To compress 3-D data, there are three steps. First, encode the data using a 3-D wavelet transform. Second, an entropy constraint quantization eliminates transformed data below a certain threshold. Third, code the quantized data using run-length coding followed by Huffman coding to reduce the amount of data. The decompression process simply reverses these steps. Alternatively, other quantization methods have been researched and shown to give even better results, such as zero-tree encoding [Pratt96].

Which wavelet to use must be chosen by the designer. On the one hand, a short filter reduces the amount of computations. On the other hand, a longer filter gives a better image reconstruction. Szu and Hsu show that a 7 and 9 biorthogonal wavelet gives a much better reconstructed image than the 2 coefficient Haar wavelet, using the same compression ratio [Szu97]. In order to achieve the same compression ratio, the quantizer had to throw out more information for the Haar wavelet. This results in a visibly less appealing image after the decompression.

Lewis and Knowles apply a 3-D DWT to video data for compression [Lewis90]. The authors combine knowledge of the human visual system (HVS) and image redundancy to achieve higher compression without noticeably affecting the video quality. Current methods use the DCT, but the computational complexity of the DCT requires that it be implemented in small blocks, which introduces blocking effects. Since it has no blocking effects, the wavelet transform outperforms the DCT method. Another reason to choose the DWT is that it has better frequency band separation than other subband coding transforms. To keep the computational complexity of the DWT minimal, short filters can be used. For example, Lewis and Knowles use filters with 4 taps for the spatial

dimensions. Also, computing the DWT can be done in parallel as well as pipelined. Therefore, the 3-D DWT shows promise for video applications.

Knowledge of the human visual system (HVS) can be used in conjunction with video compression. The HVS recognizes edges in images, but the human eye tends to see false edge intensity values. An edge appears in all octaves of the DWT. Using this information, the compression procedure can weed out insignificant data. The first compression algorithm starts by generating a 2-D DWT of an image. The algorithm keeps the lowest octave data. The higher octave's data are kept when the corresponding data from the NEXT LOWER octave exceeds a threshold value, i.e. it contains edge information. This process loses fine textured information. For a 17:1 compression rate, this algorithm gives satisfactory results, though the loss of texture can be seen.

Next Lewis and Knowles perform a 3-D DWT compression. They begin with the 2-D DWT for 3 octaves, which they apply to every frame in a video. The 4 tap Daubechies wavelet is used on the spatial domains, since it gives pleasing results in images. Once the 2-D DWT is complete, a 1-D DWT is performed along the time axis. They chose a Haar wavelet for the third dimension, since the Haar wavelet has the shortest filter length possible at only 2 taps. A short filter length means that the latency stays low. Since the experiment transmitted the decomposed video, the extra processing resulted in a delay of 1/4 second. The other wavelets tried did not improve the results, though the latency increased. Similar to the 2-D compression, this 3-D compression keeps the coefficients only when the coefficient in the above octave has a non-zero value.

The authors claim that a 3-D DWT compression ratio of 78:1 cannot be easily distinguished from the original [Lewis90]. A higher compression ratio of 156:1 still gives good results, and the authors think this can be pushed even higher. Note that the data was kept 8 bits wide throughout the experiment. This algorithm would be easier to build onto a chip than one using motion vectors. The

authors concede, however, that they do not know how well this method would compare to the DCT with motion vector algorithm. This method should be easier to implement in hardware, but a comparison to the DCT algorithm with motion vectors needs to be done first.

Goh, et al. reports that the 3-D wavelet transform developed by Lewis and Knowles is very sensitive to the threshold setting, and does not do a good job with texture content in images [Goh93]. They present a new algorithm with these points in mind. Like the algorithm of Lewis and Knowles [Lewis90], this compression algorithm performs a 2-D wavelet transform on each frame, followed by a 1-D Haar wavelet transform along the time axis. Due to the nature of human vision, Goh, et al. uses a large value for quantization of high frequency components and a smaller value for low frequency components. This is because the eye does not perceive high frequency intensity changes accurately. The algorithm decides which data values to keep before using run-length coding.

Stage 1 of the coding algorithm is as follows. The algorithm takes the 2-D WT coefficients for a block of 16 images and forms an average image. Called the temporal DC subsequence, this average image contains information on texture and edges. First, the data values of the temporal DC subsequence produced by the first octave low-low filters are all kept. Next, the data values from the other first octave filters are kept depending on the comparison to the spatial threshold. Instead of comparing the coefficients from the high-high (HH), High-Low (HL), and Low-High (LH) filters (from the first octave filters) to the spatial threshold, the algorithm compares the coefficients from the second octave filters, since edge information appears in both resolutions. The data values that are not kept are set to zero.

Stage 2 of the coding algorithm compares the low-low (LL) pass filter coefficients of the third octave to the temporal threshold for all the temporal images. In other words, it decides which coefficients to keep along the time axis. Like the comparisons in stage 1, it uses the coefficients from

the next temporal level to decide which to keep.

The third algorithm stage looks at the next lowest levels of the coefficients remaining and compares them to the spatial and temporal thresholds. Using higher threshold levels for the first octave's coefficients gives a better compression ratio. After the third stage of the algorithm, the "kept" coefficients are run-length encoded.

Tests of this algorithm were performed on the "Miss America" and "Girl on the Phone" video clips. The "Miss America" video had a PSNR of 38.09 dB with a compression ratio of 158.5 [Goh93]. To achieve even better compression, Goh, et al. codes the differences between two successive temporal DC subsequences. Such technique works due to the nature of these video clips, since there is little movement, and the background does not change, much like video conferencing. Coding the differences in this way allows this algorithm to outperform the CCITT H.261 standard. The "Girl on the Phone" video originally had a PSNR of 34.39 dB with a compression ratio of 46.1. Using difference coding, the PSNR remained the same but the compression ratio increased to 57.2.

Szu and Hsu address the problem of N-to-N teleconferencing [Szu97]. They propose a wavelet based compression system, combined with motion estimation. Such a solution for teleconferencing must be affordable and able to deal with limited bandwidth. The users of teleconferencing know a good system when they see one. People expect the video to be free of artifacts, have good resolution, have low latency, and for the audio to correspond with the video. All of these expectations add to the user's perception of quality. The proposed teleconferencing system addresses the latency and resolution concerns [Szu97].

Video transmissions can be combined with subband coding, which is related to the DWT. Subband coding generates an approximate signal and a detail signal, each one half the size of the original (assuming two subbands). A wavelet decomposition unit cascades subband filters to

produce a multi-resolution analysis. Limited bandwidth demands that a low resolution image (approximate) be sent first, followed by high resolution (detail) signals as the bandwidth allows [Szu97].

In an N-to-N teleconference, each terminal must have its own video coder/decoder (codec). The codec will perform the DWT, the inverse DWT, motion estimation, quantization, and run-length and Huffman coding. Also, the network requires a clock for synchronizing and assigning bandwidth to the terminals. Options can be built in to the design. For example, the user can choose the resolution and whether the video should be color or greyscale. The designer can leave the wavelet to be used as an option, as well as the compression ratio [Szu97].

Szu and Hsu apply their codec to the "Miss America" video sequence, with 256 x 256 resolution, and 256 greyscale values. They show how the 7 and 9 biorthogonal wavelet compares to the 2 coefficient Haar wavelet. The 7 and 9 biorthogonal wavelet has a higher overall compression ratio, and a higher PSNR for each of the 21 images in the video. The 7 and 9 biorthogonal wavelet clearly outperforms the Haar wavelet [Szu97].

3. DWT ARCHITECTURES

3.1 INTRODUCTION

The discrete wavelet transform can be implemented both conceptually and physically with filters. The transversal filter is the simplest to build. The theory uses the Z-transform in the filter's drawing. Z^{-1} indicates a delay of one unit between input signal values, and the number of delays gives the order of the filter. This filter is an example of a finite impulse response (FIR) filter, which means that it is not recursive. The outputs of a FIR filter depend only on current and previous inputs, not on previous output values. The inputs (X) flow from left to right and are multiplied by the filter coefficients (A), and the results are added together to form the output (Y), figure 3.1. Note that input X_0 is the current input, while X_1 is the previous input.

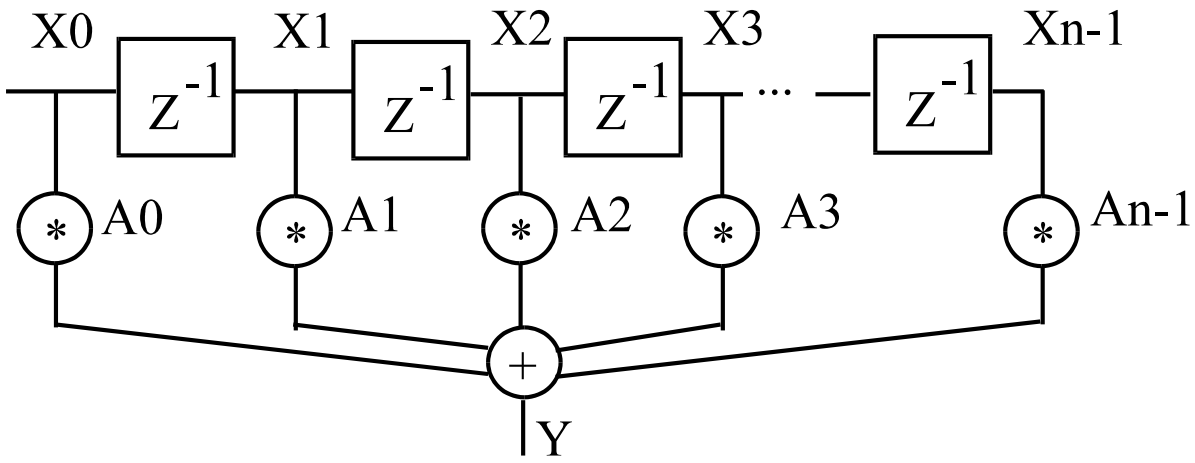


Figure 3.1 - An example FIR filter [Quarmby85]

Suppose the input signal consists of all zeroes except for a single exception of x0=1. The

filter responds to this signal with the output sequence A0, A1, ... A(n-1), 0, 0, ... 0. The output for this particular signal has the special name 'impulse response', and as the outputs show, it has a finite duration of non-zero values. The impulse response, in wavelet literature, shows what the wavelet function looks like. Thus figure 3.1 demonstrates a finite impulse response filter. This filter can also be built in a lattice version [Quarmby85].

Another type of filter is the Infinite Impulse Response (IIR) filter, though seldom if ever used for the DWT. As the name suggests, these filters use feedback, and continue to produce a response. "QMF filters can be designed by using FIRs or IIRs" [Gonzalez96]. But, FIRs are preferred because they are stable, and they have no phase-distortion.

DWT filters can be implemented either in serial or parallel. The basic serial filter design uses L multiply and accumulate cells (MACs), where L wavelet coefficients are used (L is the width of the filter). In a parallel filter, the inputs come to the multipliers simultaneously. The parallel filter has L multipliers, one for each wavelet coefficient. The multiplications are done simultaneously, then their outputs feed to a tree of L-1 adders to compute the results in the least amount of time [Chakrabarti95]. The adder tree has a latency of: (time needed for one addition)*log(L). Adding the time needed for the multipliers to this gives the filter latency.

Most architectures are digital, but Gonzalez, et al. presents an alternative for CMOS that is analog. "VLSI implementation of Wavelet Transformation techniques can be digital or analog; the later being either continuous-time or sampled-data; i.e. Continuous Wavelet Transform (CWT), or sampled-data, Sampled-Data Wavelet Transform (SDWT) respectively. Bandwidths in the analog implementations cover the audio spectrum; while sampling rates of up to 20 MHz are reported in totally digital systems" [Gonzalez96]. However, because of mismatch errors between digital and analog CMOS structures, the precision is limited to approximately 10 bits. The FIR filters designed

by Gonzalez, et al. are small, which means it uses less power. However, the reconstruction is not perfect. Due to these problems, only digital designs will be considered here.

While the DWT can be implemented with software or pre-existing sets of chips, dedicated single chip architectures will give the best performance.

How many wavelet coefficients should be used? This is a parameter of the application. Fridman and Manolakos point out that "the number of filter taps L is usually in the range 4 to 12" [Fridman97]. All of the architectures described here can be implemented on a single-chip. The size and type of the wavelet filters do not matter since most of the architectures are scalable.

We consider the low pass computations first. The high pass calculations can be done either by doubling the multiply-accumulate hardware in the PEs, or by doubling the clock frequency and having each PE do two computations per original clock period. Thus, the high pass filter can be multiplexed in space or time [Fridman94a]. Examples of time multiplexing can be found in Syed [Syed95] and Limqueco's [Limqueco96a] work. The first architecture of Knowles [Knowles90] shows space multiplexing. "This architecture can be modified by introducing an additional register for holding High Pass wavelet filter coefficient so that Details of the signal can also be computed by the same set of PE's during the time the PE's are idle. To achieve this operation, the input signal should be retained by each PE for an additional clock cycle such that the signal is available for computing the Details" [Syed95].

"The process of executing many operations in one processor is referred [to] as folding. The two main drawbacks of the folded architecture [are] the long interconnection wires from the output converter to the filter modules and functions are under-utilized. Though, this architecture has the advantage of low latency, arbitrary wordlength and not restricted to multiple of 2^m for m resolution wavelet realization" [Chen95]. Folded architectures increase the processor utilization. Categories of

folded architectures are serial, parallel, and serial-parallel (for the 2-D case), depending on the manner which the filter requires the inputs. In a folded architecture, a pair of filters is used, and the output from the low-pass filter feeds back to the input. This allows multiple octaves to be computed by a single pair of filters. This architecture has a couple of drawbacks, though, in that it requires storage equal to the size of the input signal, and the latency is large [Chakrabarti96].

The folded architecture uses only 1 low pass and 1 high pass filter to deconstruct a 1-D signal. Folding, also called multiplexing, means that one computation unit does multiple calculations. But the DWT requires a special folding algorithm, since the octaves have different amounts of calculations. For example, if the first octave does N computations, then the second octave only does N/2 computations. Most folding algorithms assume a constant number of calculations, and are thus not appropriate for the DWT. The design of the folding architecture uses "lifetime analysis" to figure out the minimum number of registers which the design should include. The lifetime analysis algorithm by Parhi and Nishitani [Parhi93] appears below:

1. Look at one period of one octave of computations
2. Write down a schedule for the computations
3. Create a lifetime chart for the variables in step 2
4. Find the minimum number of registers from the lifetime chart
5. Allocate the registers using the forward-backward plan

A lifetime chart shows the maximum number of variables in use ("live") during a period. The maximum number of live variables represents the minimum number of registers needed by the converter.

A digit-serial architecture (also called digit pipelining) uses the input digits one after the other, and the outputs are produced similarly. The arithmetic components are smaller, but are not as

fast. Digit-serial architectures have a high utilization of hardware, with less routing. Digits are smaller than words. For the first octave, the digits are half the size of the input words. A data converter breaks the input words into 2 digits. The converter stage increases the latency. Similar to the digit-serial approach are short-length filtering algorithms. Short-length algorithms try to eliminate calculations by taking advantage of calculation redundancy.

The digit-serial design [Parhi93] works with a certain amount of bits per cycle, known as the digit-size. To classify as digit-serial, the digit-size must be more than one bit, but less than the full word-length. In other words, the design is between a bit-serial and a bit-parallel design. The DWT digit-size varies with the octave j. The j^{th} octave processes wordlength/2^j bits at a time. Naturally, the design requires a data format converter to reconcile the constant clock period with the varying data rate. As before, lifetime analysis allows the designer to minimize the number of registers needed in the data converters. Due to the downsampling implicit with the DWT, the designer can view the computations as additions of even and odd parts. Therefore, the architecture can send out the even part of the output followed by the odd part in order to keep the data rate consistent.

The digit-serial design has positive and negative features. It allows the designer to use local interconnections, which reduces routing. Also, one clock operates the entire design. It utilizes the hardware 100%. The lower octaves need less supply voltage because of the smaller digit sizes, which lowers the overall power consumption. However, the increased amount of processing, i.e. the data conversion, increases the latency. Also, the wordlength must be a multiple of 2^J, where J represents the number of octaves [Parhi93].

Both the folded and the digit-serial architectures implement the 1-D DWT, and can be used to implement the 2-D DWT as well. The folded architecture is faster, but larger. It has low latency and allows for arbitrary wordlength. The digit-serial architecture uses less power and has simpler

interconnections, but the speed and wordlength are constrained. Each architecture requires the same amount of input/output pins. With these features in mind, a project designer can choose which of these architectures best suits the target application.

Finally, quadrature mirror filtering (QMF) is a variation which has only half of the MACs required for a direct implementation. But QMF needs complex control and increased routing to allow for fewer multipliers. "QMF is a special case of subband filtering, also called subband coding. A bandlimited signal can be filtered into its low-pass and high-pass components. Each of these components represents half of the spectrum, or a subset of the original signal bandwidth, thus the term subband. If the analyzing filter coefficients are symmetric, then the synthesizing components are mirrored with respect to the half-band value, thus the term quadrature mirror…" [Brooks96].

3.2 ARCHITECTURE CONSIDERATIONS

Latency, control, area, storage, and precision are the main considerations in judging an architecture. An algorithm's latency can be found by subtracting the last time step that does a calculation from the first calculation time step. In other words, the latency measures how fast the algorithm solves a problem. Naturally, a large input will take longer to compute than a smaller one, so the latency is expressed as a function of the input size. The lowest possible latency is when the calculations are done as fast as the input rate.

Control signals specify timing operations. There are several types of control: centralized, flow, and prestored. A centralized controller generates the signals in one spot and distributes them to the other components directly. Flow control sends the control signals (tags) from PE to PE along with the partially computed data. A designer might also build the control into the PE's, called

prestored control. Centralized control may be easier to implement, but may impact the scalability and increase the area, so the choice of control is important to a good design.

Systolic filters demonstrate control and latency issues. The filters contain processing elements (PEs), and the processing elements contain multiplexors. Based on a timing signal (the clock) the multiplexors switch between 2 or more inputs. A fully systolic design interconnects between the nearest neighbor cells only. Though the cells become more complex than in a semi-systolic design, the regularity and extensibility are kept. Array architectures are for distributed memory and control. This allows the PE's designed for one case to be slightly modified or arranged to solve another case. The arrays that Fridman and Manolakos derive are systolic and semi-systolic, due to the communication lines. The semi-systolic 1-D DWT processing array meets the lowest possible latency, while the fully systolic array has a latency of 2N-1.

Vishwanath, Owens and Irwin [Vishwanath92] compare three DWT architectures based on area, latency, and period. The designs are meant for VLSI implementations. The authors base their architectures on the linear systolic array. The systolic method for the DWT has a lower bound of: Area * Latency^2 = Omega(input length^2 * filter length * precision rate^3 / input-output rate^2). The Inverse DWT case is ignored since it is so similar to the DWT.

Several types of storage units are available such as MUX-based, systolic, RAM, reduced storage, and distributed. The MUX-based storage unit has an array of storage cells that form serial-in-parallel-out queues. Multiplexors decide which cell values are sent to the filters. This approach is regular, but it is not extensible. The semi-systolic storage unit is similar to the MUX-based unit, except that busses and tri-state buffers are used in place of multiplexors. This allows more octaves to be added without changing the multiplexor's size, and it keeps the regularity. The semi-systolic unit has long busses, which is a disadvantage. In the RAM based storage unit, a large RAM replaces all

of the storage cells. The RAM has the advantage of being compact, but it requires addresses to choose the appropriate datum. Allowing for more octaves means that the RAM size and address decoders will need to be changed, so this design is not as scaleable. The reduced-storage unit uses the forward-backward allocation scheme to move data through the unit, allowing for a minimum of storage cells. The minimized storage is a trade off for more complex control and less regularity and scaleability. The distributed architecture has local storage on each processor in the filter, which is analogous to parallel computers. The processors have a storage size equal to the number octaves. Additionally, each processor needs to have multiplexors and demultiplexors in order to move data to other processor's memory. These storage unit architectures each have advantages which make them more suitable than the others depending on the implementation selected [Chakrabarti96].

The input data and output data precision are important to a signal processing architecture design. Typically, video applications assume an input precision of 8 bits, corresponding to 256 shades in the greyscale. However, for other applications, the input precision could be 12 bits or more corresponding to the input sampling hardware, e.g. an analog to digital converter.

The intermediate coefficients are larger than the original data, since the intermediate coefficients consist of a summation of terms multiplied by the wavelet. In other words, the range of the intermediate coefficients grows. The rate of growth depends on the wavelet, but typically this rate does not exceed 2 [Grzeszczak96]. To compute lower octaves, the intermediate coefficients are again passed through filters, which increases the range for each octave computed. For example, a data value of 8 bits will need a 9th bit after the 1st octave computation, then a 10th bit for the 2nd octave computation, and so on. Every time an intermediate coefficient passes through a filter, its range grows. This includes multidimensional wavelet transforms. For the 3-D architectures introduced in the next chapter, this means that the internal multipliers and adders will need to handle

data as wide as the outputs. The signal-to-noise ratio (SNR) measures the difference between the floating point DWT coefficients and the ones rounded off due to finite precision.

In the DWT calculation, there are not exactly N/2 filter outputs for N inputs. This is due to the fact that the filter implementation lengthens the outputs by the number of the delay stages within the filter. Edwards discusses this problem in relation to lattice filters, but it also applies to other filter types [Edwards92]. There are two ways to deal with this problem. The first is zero padding, where any value after the last input is assumed to be 0. This adds few extra coefficients, equaling the number of delay stages, but this is tiny compared to the input size. The second solution assumes that the input is circular. Circular input means that the input starts over at the beginning when it reaches the end. For 1-D data, imagine a circle. For 2-D, a torus describes the input. Both solutions allow perfect reconstruction. Zero padding is easier to implement in terms of routing, therefore, it is the default in this work.

One last consideration is the on-line versus off-line case for the inverse transform, which depends upon the desired application. The Inverse Discrete Wavelet Transform (IDWT) architectures come in two forms. One assumes that all of the DWT outputs are stored in memory (the off-line case), while the second assumes that the DWT analysis happens right before the IDWT synthesis (the on-line case). Latency concerns are important in both cases. The latter case requires a buffer of size $2*2^J$ between the DWT and IDWT, where J is the number of octaves. Due to the similarity between the DWT and IDWT, the same hardware can compute both functions, with a few modifications. Some applications, such as the FBI fingerprint compression project [Bruce96], only need one specific wavelet filter on the wavelet analysis/synthesis chip. Therefore, the programmable filters can be replaced by fixed filters, which reduce the area required.

3.3 ARCHITECTURES FOR THE 1-D DWT

Several algorithms have been developed to compute the DWT based on the pyramid algorithm (PA), also called the direct pyramid algorithm (DPA) [Mallat89]. The focus is on the algorithms as they are implemented in hardware.

The recursive pyramid algorithm (RPA) [Chakrabarti96] is a variation of the PA where a filter's outputs are fed back into the filter to compute lower octaves. The RPA was devised to reduce the storage and latency requirements of the folded architecture. It interleaves the octave outputs such that the first octave outputs come every other cycle. There is good reason for staggering the outputs like this [Fridman97]. The lower octave outputs are generated when no higher-octave outputs are available. A modified recursive pyramid algorithm (MRPA) [Chakrabarti96] exists to schedule pipelined parallel filters, which includes a shift to deal with the filter's latency. To reconstruct the signal from the filter outputs, one can use the inverse recursive pyramid algorithm (IRPA) [Chakrabarti96].

Fridman and Manolakos present a scheduling algorithm and architecture for computing the 1-D DWT of J octaves [Fridman94a]. The accompanying architecture could be implemented with multiple processors instead of a VLSI layout, but the algorithm's schedule is most significant. The authors go in-depth in their data dependency and localization analysis of the 1-D DWT. Multiprojection maps the 3-dimensional algorithm onto 1-dimensional arrays. Note that they are talking about variables with a 3-D index space, rather than the 3-D DWT. The DWT has variable length data dependence vectors, therefore it "cannot be linearly mapped onto an efficient regular architecture by conventional space-time methods" [Fridman97]. The authors reformulate the DWT in order to use linear mapping methods.

The basic 1-D DWT algorithm has a nested loop with three index variables. The first

variable represents the octave number, called j. The second variable keeps track of the data sample, denoted by n. The third variable, k, indicates which wavelet coefficient to use. Inside the innermost loop, the algorithm performs 2 multiply and accumulate instructions, one for the high pass filter and one for the low pass filter. This algorithm is simple and straightforward, but the three index variables present a problem. This means that the intermediate transform values are a function of a 3-dimensional index space. Also, the inter-octave dependencies are not regular, but Fridman and Manolakos [Fridman94a] transform the algorithm to improve this. The second algorithm change maps the 3 index variables onto 2 by eliminating the j variable. In essence, this change skips every other time slot between the first octave computations, allowing the computations of higher octaves to fill in between. The higher octaves also skip every other time slot so that all octaves can be computed. For example, this forms the pattern "1 2 1 3 1 2 1 4 1 2 1 3 1 2 1..." for the 4 octave case, where each number represents a computation done for that octave.

This schedule allows for minimum latency, and is efficient regardless of the input length, or number of octaves. The authors introduce a non-linear transformation of the index space to make a linear, regular array. Another algorithm transformation makes the index space two-dimensional, which avoids data collisions. The final algorithm results in a design with L processing elements (PEs), where L represents the filter length. The design computes a 1-D, J-octave DWT with latency equal to the input signal length [Fridman94a]. See figure 3.2 (a) below.

To implement their algorithm, Fridman and Manolakos devised a PE array, and showed it for the 4 octave DWT. The latency for this design is the same as the input length. It has an efficiency of $1 - 2^{-J}$, which gets closer to 100% as the number of octaves increase. Each PE needs memory registers, depending on the total number of octaves. This design is simple and modular, with distributed control. Due to the scalability of the design, it should translate to 2-D [Fridman94a], as

shown in the architecture of Chen [Chen95].

Fridman and Manolakos present a 1-D DWT processing array of three octaves, with a latency of 3N/2, where N denotes the input length [Fridman94b], figure 3.2(b). Three octaves are optimal for this design, with 58% utilization. Having four or more octaves increases the latency, due to data collisions of the scheduling. The authors want a scalable, regular VLSI architecture based on systematic analysis methods. The PEs built to perform the DWT should be simple and have distributed memory and control. The PEs of this architecture have 5 memory registers each. The design needs L PEs, where L designates the number of wavelet coefficients.

Syed, et al. developed a systolic design for the 1-D DWT, and noted that only half of the PEs do calculations with useful results [Syed95]. The other PEs stay idle due to the downsampling operation. To improve hardware utilization, the authors added a register for the high-pass coefficient. During their idle time, the PEs will calculate the details. The input signal must be kept for an additional clock cycle, but this change increases the hardware utilization and eliminates the need for a high-pass filter. In effect, this architecture maps the high-pass computations in time instead of space. Figure 3.3 shows the PE array for a 3 octave DWT. With a slight modification, this architecture can be used to compute the inverse DWT as well.

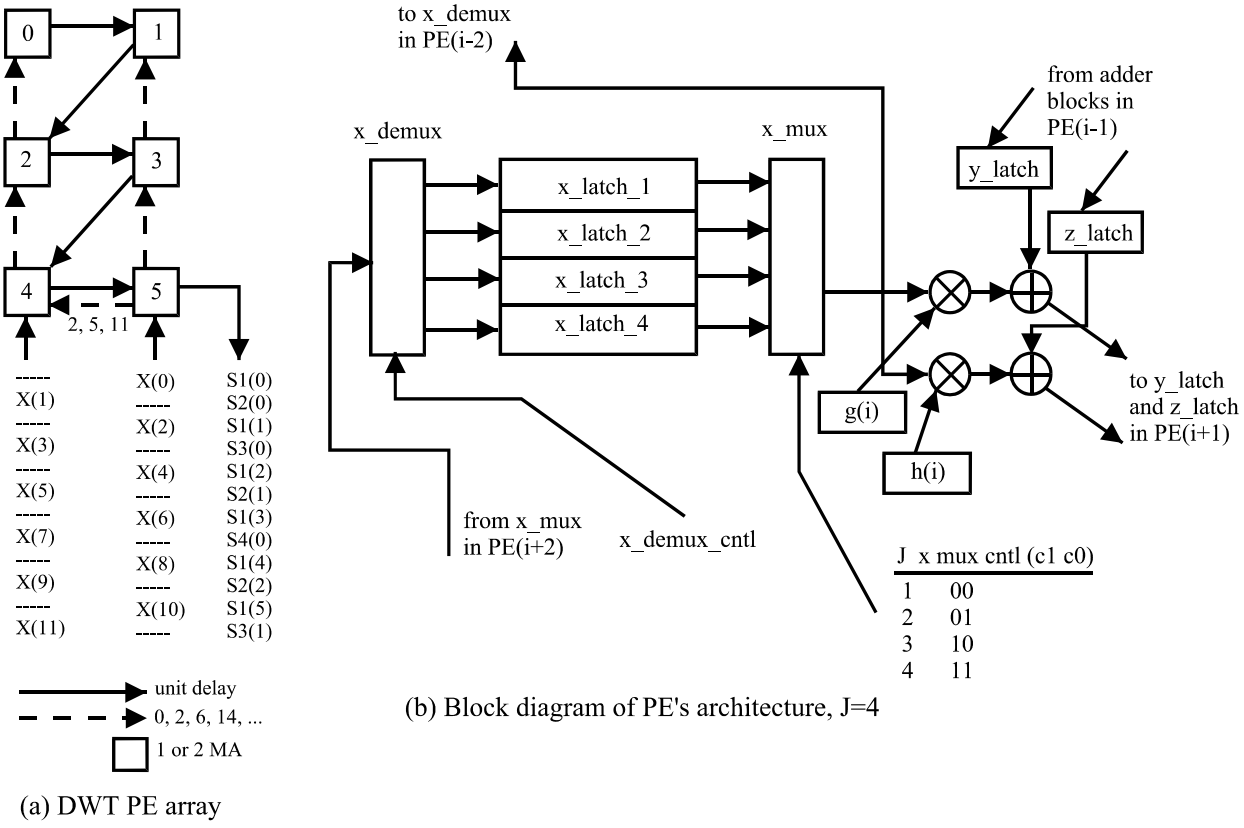


Figure 3.2 - Fridman's DWT PE array

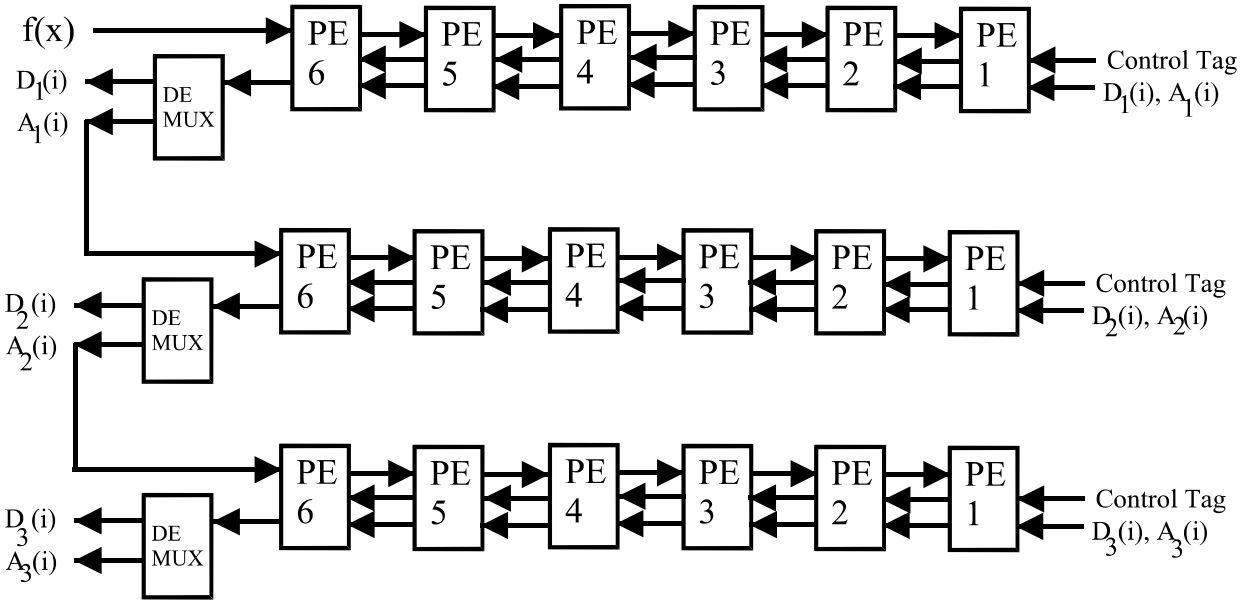


Figure 3.3 - Architecture to compute 1-D DWT by Syed et al.

Knowles presents a fully pipelined architecture for the 1-D DWT [Knowles90], figure 3.4. It uses a low pass and a high pass filter. Serial-in-parallel-out (SIPO) shift registers store the filter inputs. One stores data from the input stream, while the others store data generated from low pass filters. Thus, a design with J octaves will require J shift registers. The shift registers should be as deep as the longest filter. Because of the scheduling, the contents of each shift register will be used right after it fills up, but before it needs to store another value. The shift registers connect to a multiplexor, which sends the contents of the correct shift register to the filter pair. Both filters get the same inputs. They perform the convolution in parallel, and pass along the results. The high pass filter's output goes to a demultiplexor. It decides which channel to send out the output to, based on the octave. Another demultiplexor routes the low pass filter's output back to the appropriate shift register. A small control unit generates the signals to run the multiplexor, demultiplexors, and when the shift registers should shift. Together, these parts form a 1-D DWT processor.

Knowles implemented this architecture in ELLA, a hardware description language. Results show that making this circuit with 4 fixed coefficients and 3 octaves would require a NEC CMOS5 gate array with 1500 gates. It would be able to run at 6 MHz at least. This shows that a high-speed DWT chip can be made with a small amount of gates.

Aware Inc. introduced a wavelet transform processor [Aware94], which allows up to 6 coefficients. The user chooses the wavelet coefficients, either specifying the coefficient values or the pre-loaded 6 coefficient Daubechies transform. Input and output data can have a width of 16 bits, and can be fed at a rate of 30 MHz. The chip processes both the DWT and the IDWT. A delay of 9 clock cycles occurs between the time that the first input is received and the time that the first output

is complete.

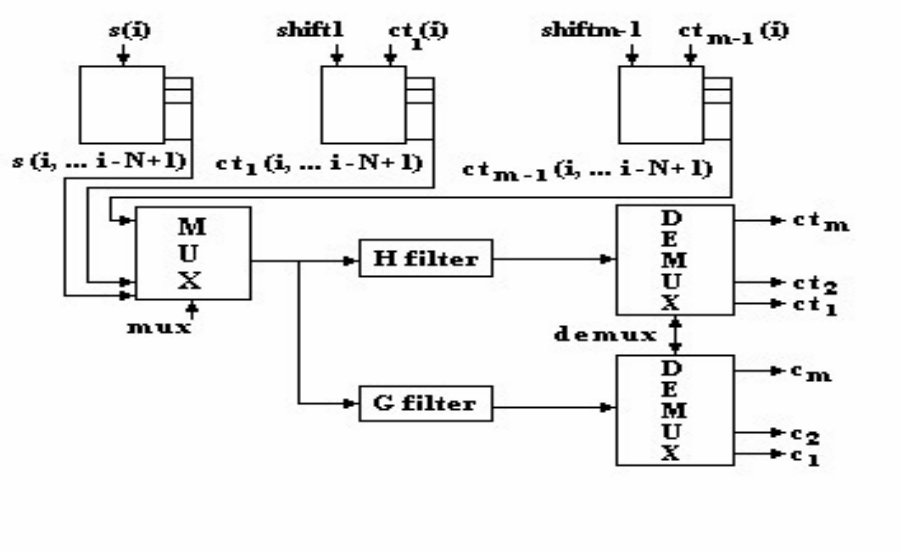


Figure 3.4 - Knowles' 1-D pipelined architecture

The Aware architecture is pipelined into 4 stages, figure 3.5. Each stage has registers, a multiplier, adder, and shifter. Though the multiplier is size 16 x 16, only 16 bits are passed on from the multiplier. Software chooses which 16 bits of the result to keep. Two 16 bit, bi-directional busses allow inputs and outputs to the four stages, depending on the cross-bar switch, while one output bus always supplies the outputs. A designer could cascade a number of these chips to achieve a longer wavelet.

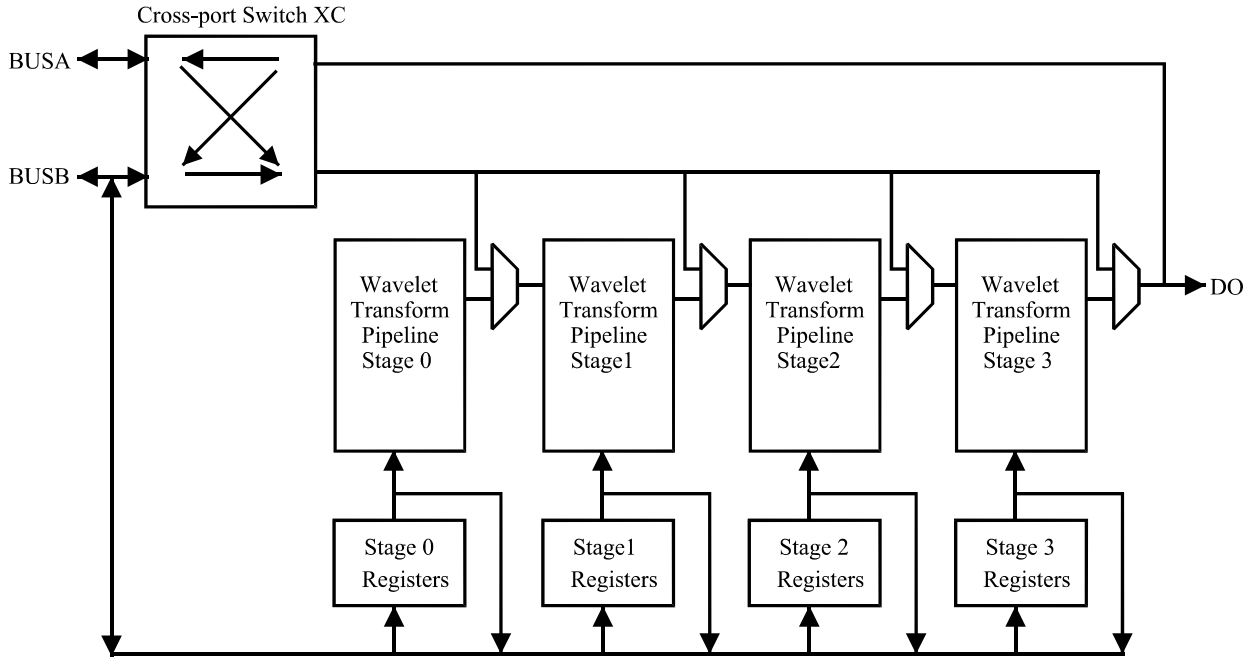


Figure 3.5 - Aware's 1-D architecture

Parhi and Nishitani present two architectures for the DWT [Parhi93], figures 3.6 and 3.7. Their paper examines the folded wavelet architecture first. Regardless of how many octaves of decomposition are required, the design only needs 1 low pass and 1 high pass filter. The second architecture, digit-serial, computes each octave with different sized processors.

Parhi and Nishitani assumed a filter with 4 taps. A wavelet with more coefficients requires more registers, so the application's needs affect the area and latency of the final design. The authors also assume 2 pipeline stages within the design. Another design consideration deals with handling intermediate results. To juggle them, one should add an output converter unit. For example, an output from octave 1's low pass filter will be used as input to both the low and high pass filters of octave 2. Finally, carry ripple adders in the design affect the overall speed, although Parhi and Nishitani note that the speed would still be practical for video applications [Parhi93].

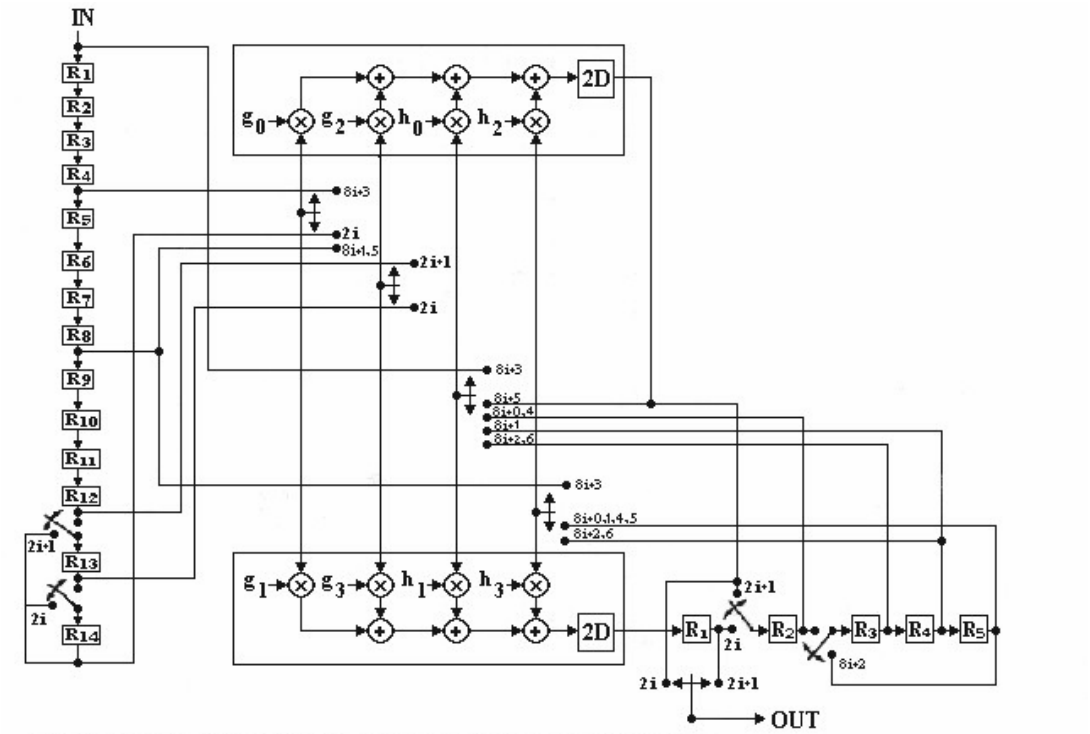
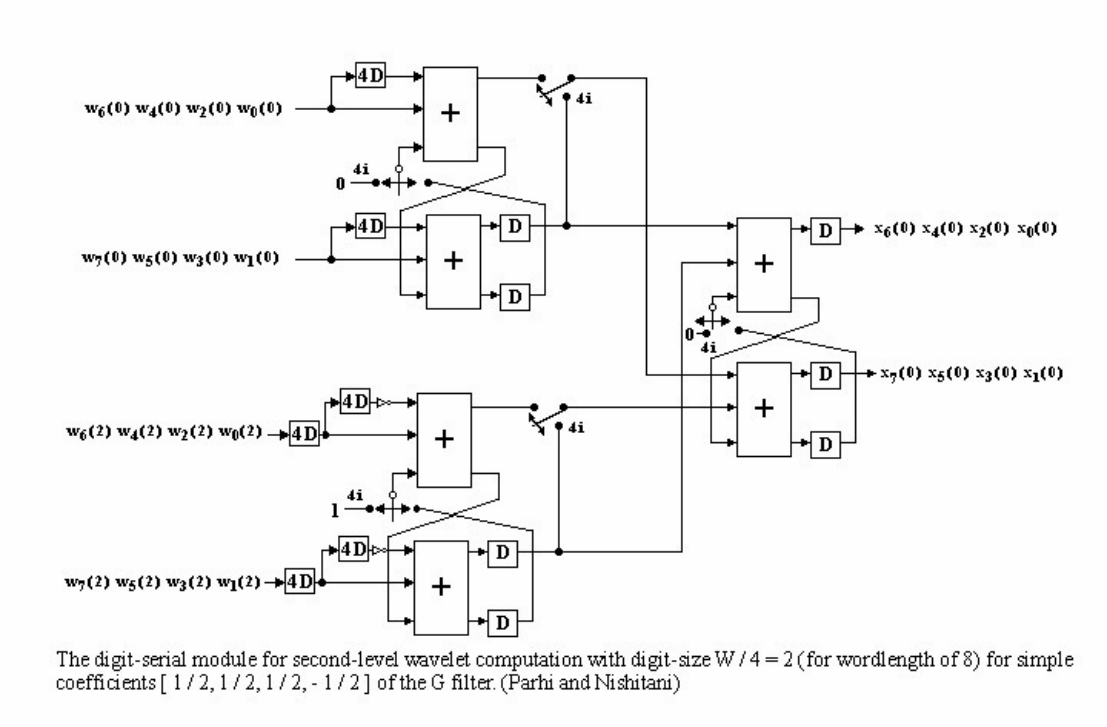


Figure 3.6 - Parhi and Nishitani's folded 1-D architecture



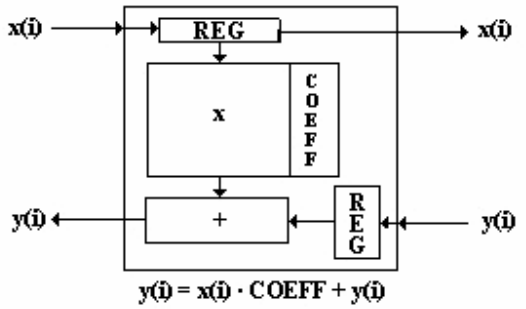
The digit-serial module for second-level wavelet computation with digit-size W / 4 = 2 (for wordlength of 8) for simple coefficients [1 / 2, 1 / 2, 1 / 2, - 1 / 2] of the G filter. (Parhi and Nishitani)

Figure 3.7 - Parhi and Nishitani's digit-serial 1-D architecture

Vishwanath, et al. [Vishwanath92] developed three architectures for the 1-D DWT. Their first architecture cascades linear systolic arrays in a matrix, where each row computes one octave, while each column contains a multiply and accumulate cell (MAC) for each wavelet coefficient. Figure 3.8 shows this architecture. The input flows from left to right, while the output flows in the opposite direction. The left-most cell receives the input for its row, and passes the output from its row to the row beneath it. One output can be created every two clock cycles, since the cells include downsampling. Due to this timing issue, two overlapped input streams can feed into this architecture, and two DWTs can be computed simultaneously. Each row uses half as many inputs as the row above it, and it produces half as many outputs as well, but has the same timing as the row above. To accommodate for the latency mismatch, the registers within the cells of the lower octaves (rows) must be made larger.

The above discussion assumes only one type of filter is needed. A practical application has the architecture computing the low and high-pass outputs for a signal. Instead of computing one type of filter on two inputs, it would make more sense to modify the cells to compute two types of filters for one input stream. The designer simply adds another coefficient to each cell, and adjusts the cell's multiplier to alternate between coefficients. This technique, called time mapping (as opposed to space mapping), also works for the other 2 architectures of Vishwanath, et al. This first architecture suffers from a large area requirement, and keeps the processors busy only 33% of the time.

The second architecture by Vishwanath, et al., figure 3.9, improves these two criteria. The improved architecture schedules the first octave outputs at every other clock cycle. The second octave outputs occur with a spacing of 4 cycles between them. Outputs in the third octave have 8 clock cycles in between. This output pattern allows for all octave results to be scheduled, so the processor scheduling maps the octaves in time instead of space (rows). The time mapping allows for savings in area, without a significant delay increase. To implement this design, a routing network is used consisting of shift registers. The network has a size of (number of wavelet coefficients)*(number of octaves). A bound of the area was calculated to be O(network size * precision). The time needed to find a DWT with this architecture is 2N cycles.



A Basic Cell

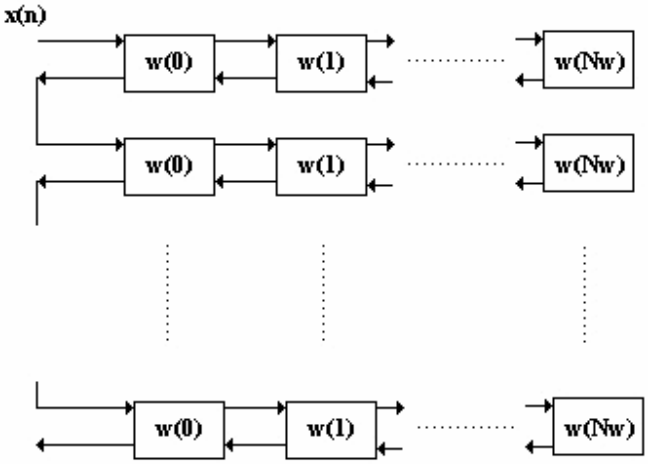


Figure 3.8 - Vishwanath's first 1-D architecture

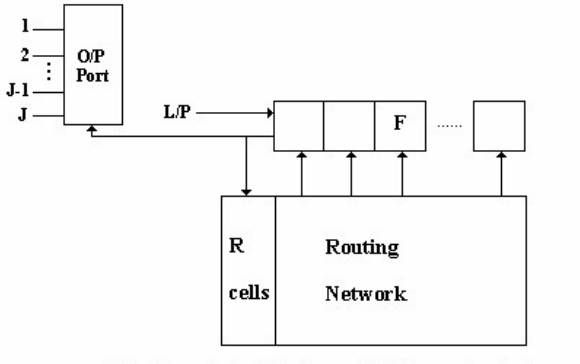


Figure 3.9 - Vishwanath's second 1-D architecture

Finally, Vishwanath, et al. proposes an optimal DWT design that has minimal area, figure 3.10. However, it has not been implemented yet because the control is very complex. The hardware would schedule the inputs on-line, depending on whether the clock cycle is even or odd. Tags specifying the octaves are added to the data. It multiplies data only when the input tag is 1 less than the output tag. Due to the scheduling difference, the basic cell for this architecture had to be modified. This architecture is not truly systolic, though it is nearly systolic. The complexity of this design means that the clock cycle must be slower than the clocks in the other architectures. The three architectures are summarized in table 3.1 below. L shows the filter length.

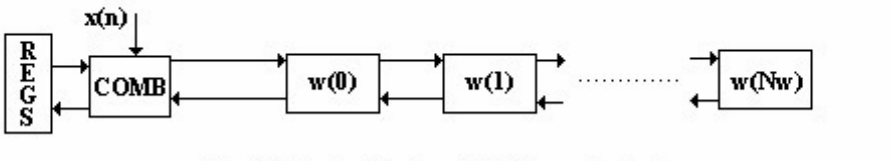


Figure 3.10 - Vishwanath's third 1-D architecture

Architecture	Area	Latency	Period
First	O(L*k*logN)	2*N+logN-4	2*N
Second	O(L*k*logN)	2*N	2*N
Third	O(L*k)	O(N)	O(N)

Table 3.1 - The three 1-D architectures of Vishwanath, Owens, and Irwin [Vishwanath92]

A lower latency bound of O(N) is important, when it can be achieved, as do Vishwanath, Parhi, and Knowles. Aware's prototype is O(N logN). Fridman's first architecture has a latency of O(N), though using time multiplexing instead of space multiplexing will double this latency. It requires L or 2L Multiply-Accumulate units, respectively.

Chakrabarti has a parallel filter architecture with N latency, not based on routing network. Vishwanath, et al. have global routing networks, which varies the amount of memory, according to table 3.2 below.

1.1	fully systolic	2L(J+4) latches	2N latency
1.2	semi-systolic	L(J+4) latches	2N latency
1.3	RAM-based	L(J+4) latches	2N latency

Table 3.2 - Vishwanath, et al. architectures

The architectures of Fridman and Vishwanath are modular and systolic. Both have a lower latency bound of N. Fridman's can be distinguished by the following observations. The filters can be lengthened by adding more processing elements, and simple changes to the PEs allow the design to do more octaves. Most importantly, the analysis can be used for arrays other than the one specifically designed by Fridman, such as 2-D designs.

Directly mapping the conceptual J-octave DWT onto hardware is inefficient, so most authors avoid this. Knowles has one of the first discrete wavelet transformers. It is not scalable, it has a large VLSI area, complex control and routing, but its latency is O(N) [Fridman97]. The architectures of Parhi, Aware, and Knowles are fast but not regular. This means that scaling these architectures for more octaves, wider filters, or to higher dimensions is difficult. Vishwanath, et al. has a fast architecture, which uses a register network with a filter pair [Vishwanath95]. Fridman and Manolakos' architecture has the advantages of distributed memory and control, following their systematic data dependence analysis [Fridman97].

3.4 ARCHITECTURES FOR THE 2-D DWT

Filtering in both the horizontal and vertical directions performs the 2-D discrete wavelet transform. The horizontal (row) inputs flow to one high-pass filter and one low-pass filter. The filter outputs are downsampled by 2, meaning that every other output is discarded. At this point, a 1-D transform would be complete for one octave. The 2-D transform sends each of these outputs to another high-pass and low-pass filter pair that operates on the columns. Outputs from these filters are again downsampled. One octave of the 2-D transform results in four signals, with each of the four signals only one-fourth the size of the original input. The algorithm sends the low-low output to the next stage in order to compute the next octave. This process repeats for all octaves.

The processing load does not distribute evenly in a parallel environment. Folding can be used to make the architecture more efficient since it requires only one pair of filters. As in the 1-D case, a folded architecture requires a large storage size. A 1-D folded architecture needs a RAM of N words, where N is the length of the input. A 2-D folded architecture, on the other hand, requires a RAM size of N*N words, since N*N outputs will need to be stored in the worst case. Besides a large storage size, the 2-D folded architecture has a long latency [Chakrabarti96].

Other architectures for the 2-D case are the Serial-Parallel, Parallel, and Block-filter architectures. The Serial-Parallel architecture has 2 serial horizontal (row) filters, and 2 parallel vertical (column) filters. A filter in this design actually contains a high-pass filter and a low-pass filter. A storage unit buffers the data sent to the parallel filters, and a second storage unit buffers the low-low-pass output before it feeds back into one of the serial filters. The first storage unit has an approximate size of 2*K*N, while the second storage unit has a size of N. The Serial-Parallel architecture lends itself to a fully parallel architecture. The Parallel architecture replaces the multipliers with programmable multipliers, which eliminates half the multipliers. Both the Serial-

Parallel and Parallel architectures are scaleable. The Block-form architecture requires that the input signal be available as a block, instead of just a row or column. The Block-form architecture needs a data converter, which reduces scalability and increases the routing. Other approaches have been proposed for the 2-D case which are not covered here [Chakrabarti96].

The DWT and IDWT are separable in the [Vishwanath94a] architecture, which means that the 2-D transform is simply an application of the 1-D DWT in the horizontal and vertical directions. They devised a 2-D algorithm that runs in real-time, and is similar to the recursive pyramid algorithm (RPA). The 1st octave has a calculation, followed by 4 cycles where the lower octaves are scheduled. This process repeats until completion. The calculations for the IDWT are scheduled similarly, with the final reconstruction getting a time slot every other cycle. The intermediate reconstructions are scheduled in between. Interleaving the octave's calculations in this manner reduce the DWT/IDWT storage requirements and overall latency. When using a vector quantizer, the outputs must be blocked, and the inverse transform must be set up to receive blocks. The blocking adds to the latency, especially in the inverse transform.

Vishwanath, et al. present the Symmetric Pyramid Algorithm (SPA) [Vishwanath94b]. The SPA and Inverse SPA (ISPA) compute the DWT and IDWT, respectively. They use the 2-D SPA to make architectures for a video encoder, decoder, and transcoder. This architecture has the advantages of minimum latency, minimum buffering, and single chip implementations for the encoder, decoder, and transcoder. The scheme is hierarchical, which means that the image can be accessed at different resolutions. Hierarchical coding applies to progressive transmissions and multi-purpose uses. [Vishwanath94a] addresses the problem of interactive multi-cast over channels with multiple rates, for example, teleconferencing over mixed networks.

The SPA architecture computes the low pass outputs with systolic arrays, and high pass

outputs use a similar systolic array. Memory stores the intermediate values, which are the low pass outputs, while the high pass outputs simply exit the SPA module. Both low and high pass filters receive the same inputs from the storage unit, which is also connected to the main input lines. The memory contains J shift-registers, which have L cells in them.

The ISPA architecture also computes the low and high pass outputs with separate systolic arrays. The architecture adds the low and high pass output together to form the ISPA output. Each ISPA filter has its own memory unit to feed it inputs. The first memory unit keeps the received DWT samples. The second memory unit stores the intermediate values. Each memory unit has J shift registers, but the length of the registers is approximately L/2 bits.

The 2-D DWT and IDWT are computed using the above SPA and ISPA modules, figure 3.13, which take N^2+N cycles. It has an area of O(NLk). The 2-D implementation needs addition memory to act as holding cells between the horizontal and vertical filters.

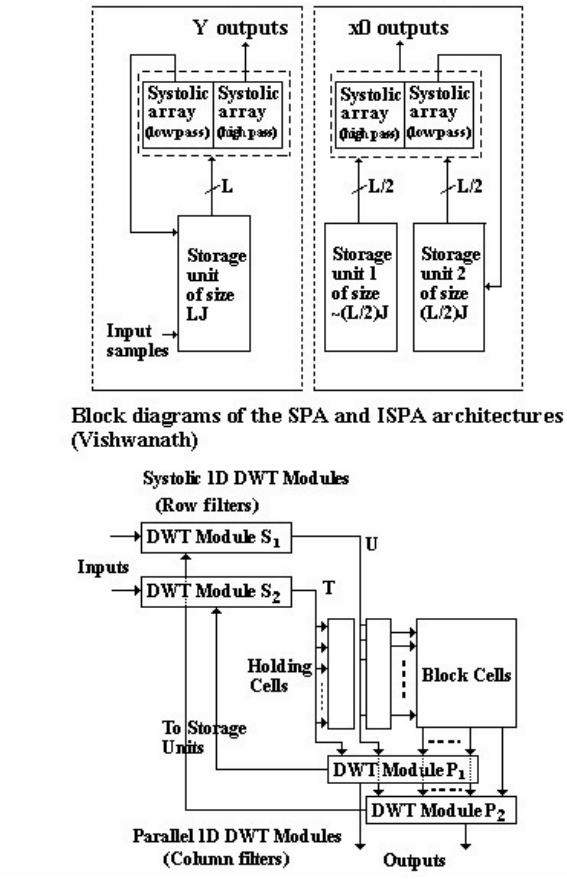
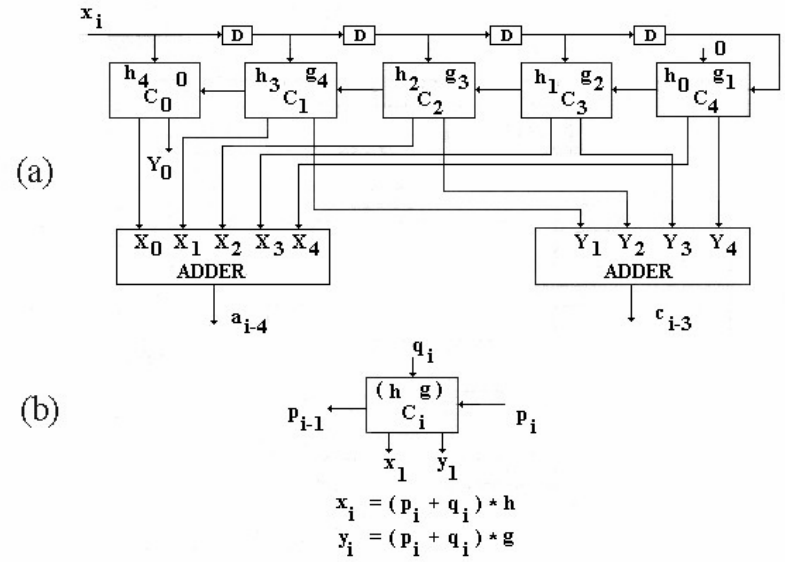


Figure 3.11 - Vishwanath's 2-D architecture

Acharya's 2-D DWT module is made from 2 1-D modules, figure 3.11. A transpose circuit is used to transfer the horizontal (row-wise) outputs to the vertical (column-wise) DWT module. The paper uses a 9-7 biorthogonal spline filter, where the low pass filter has 9 coefficients, and the high pass filter has 7 coefficients. Since the coefficients are symmetric, 2 of the low pass filter coefficients have the same value, while 3 of the high pass filter coefficients share the same value. Thus, only 5 and 4 coefficients need to be specified to the architecture [Acharya97].



(a) The Systolic Array for Computing 1D DWT
(b) The Basic Processing Element

Figure 3.12 - Acharya's systolic architecture

The first few cycles allow the inputs to load. On the following clock cycles, the architecture generates a high or a low pass filter output. On the trailing edge of an even clock cycle, a high pass filter output becomes available. The low pass filter outputs come on the trailing edge of odd clock cycles. Since the architecture generates an output for every clock cycle, the architecture is 100% utilized.

The Limqueco sequential architecture [Limqueco96a] is optimized for a 2 octave, 2-D DWT, where it has 100% utilization, figure 3.12. With modifications to the architecture, it can be expanded to include more octaves, but the utilization drops. This architecture handles downsampling

differently than other architectures. Due to the downsampling operation, DWT architectures typically either shift the input data, or eliminate every other output. The Limqueco

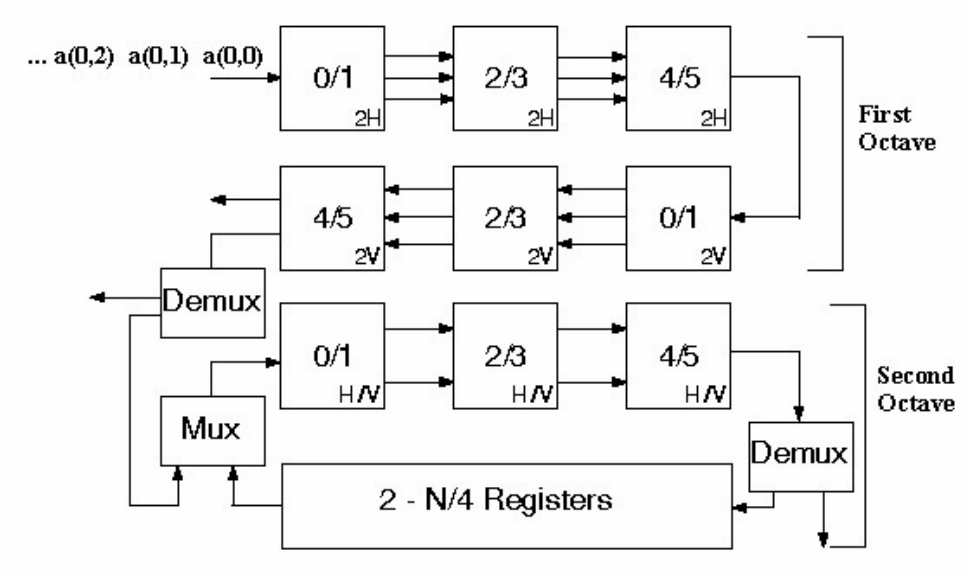


Figure 3.13 - Limqueco's 2-D architecture

architecture does neither. Instead, an even and an odd coefficient share one processing element (PE). The PE alternatively uses the even and odd coefficients, resulting in a need for only half the number of PEs, while improving the throughput by about 50%. Simply adding more PEs to the filter expands the number of filter coefficients. The filter computes the high and low pass outputs at the same time, which produces 2 outputs every other cycle. From the horizontal filter, the outputs are fed to the vertical filter one per cycle, instead of 2 outputs per every other cycle with nothing in between. Staggering the outputs this way allows the vertical filter to be utilized 100% of the time. The second octave of this architecture has only one filter. It computes both low and high pass outputs, for both horizontal and vertical directions. The second octave uses 2 register banks for

intermediate data storage.

In the Limqueco architecture, each PE has a different amount of memory, which makes scalability a problem. The architecture is fast, efficient, moderately modular, has localized wiring, and large memory is required (about twice as much). High and low calculations are done simultaneously, with 16 bit multipliers.

To expand this architecture for more than 2 octaves requires adding one more filter and removing one register bank. Though the register bank memory decreases, memory is added to the PEs. Since the PEs have different amounts of internal storage, this architecture is semi-systolic. A DWT for 3 octaves has a utilization of 91% for the Limqueco architecture, and the utilization increases for additional octaves.

3.5 OTHER ARCHITECTURES

Several proposed architectures modify the DWT algorithm to achieve specific results. These architectures are considered special cases of the DWT since they involve a non-standard algorithm, work for only one specific wavelet, or approximate the results. Thus, these architectures may not be appropriate for any given application. Aditya's design produces all octave outputs simultaneously [Aditya96]. Chang, et al. has a quick algorithm the yields an approximate solution [Chang97]. Lewis and Knowles present a way to calculate the DWT with Daubechies filters, without multipliers [Lewis91]. These designs are expanded on below.

Aditya's DWT algorithm [Aditya96] operates on the observation that convolution in the time domain is multiplication in the frequency domain. The algorithm computes the Fourier Transform (FT) of input blocks, then multiplies the transformed data with the Discrete Fourier Transform

(DFT) coefficients of the wavelet filters. The algorithm then performs the Inverse Fourier Transform (IFT) on the multiplication results. After downsampling, the resulting output streams are the DWT of the input signal. Aditya's algorithm allows for all octave information to be generated simultaneously, instead of waiting on results from the octaves above. One disadvantage is that the FT and IFT components add to the chip area. Though only one FT component is needed, the design needs one IFT component for each octave, unless making a significant sacrifice in speed. Aditya's architecture compares favorably with the other DWT designs in terms of latency, but its area is larger due to the facts that 1) the area depends directly on the number of octaves, and 2) the block length (Nl) is typically much larger than the normal filter width (Nw), since $Nl = 2^{J-1}*Nw$. For example, Nl = 7*Nw for the 4 octave case. The other architectures do not expand in area as the number of octaves increase. Also, many extra computations are done for the lower octaves. The first octave loses half of its data to downsampling. Each octave below it loses half of its data in turn. For example, the third octave throws out 7 data points for each one that it keeps. Thus, this architecture is inefficient. Martin Vetterli and Cormac Herley also had an architecture based on the FFT, only it differs from Aditya's due to recursion [Aditya96].

Chang, et al. [Chang97] presents a plan to get faster wavelet transforms at the expense of accuracy. They use fixed-point representation for simplifying data. One can think of the fixed point number as an integer with a scale factor that simply reminds one of the radix point. Treating the data values as integers greatly reduces the complexity of the hardware when designing a DWT architecture. Also, the authors propose thresholding the calculations. If the product of a data value and a wavelet coefficient will be close to zero, then the multiplication will be skipped, saving time. The hardware quantizes the two operands, and decides whether or not to perform the multiplication based on the magnitude estimate stored in a lookup table.

Chang's 1-D architecture has several notable components. A "Sliding Window" unit buffers the input data, until it is multiplied by a wavelet coefficient. The wavelet coefficients are obtained from the coefficient table, instead of being stored with the multiplier like other architectures. To perform calculations, the design has one central multiply-accumulate unit. A Booth multiplier is used, implemented with a series of shifted additions to save on hardware size. The multiply-accumulate unit contains the quantizer and threshold detector to decide whether or not to skip the multiplication. A centralized control unit generates the appropriate signals to drive the units of the transformer. Both input and output data are stored in memory, accessible through the data and address busses, and probably located off-chip. The design must alternate between computing the high pass and low pass outputs, though this is not explicitly stated. The authors do not indicate whether or not this design will fit on a single chip. Instead, it appears to be geared towards a group of chips.

Chang, et al. experimented their architecture with 1-D signals of both speech and music, which are sampled at 8 kHz and 22.5 kHz, respectively [Chang97]. Each datum has a width of 16 bits. The authors compare their wavelet transformer with a "normal" wavelet transform using criteria of PSNR and the number of multiplication and addition operations. They show that the PSNR values increase as the number of bits in the fixed-point representation increase. Also, the number of multiplications and additions decrease by up to almost 70% for speech signals, and up to almost 50% for music, depending on the decomposition level. The reduction in calculations is best for 4 or fewer octaves. The PSNR also rapidly decreases for more than 4 octaves. The PSNR varies between 65 and 90 dB for speech, and 95 and 115 dB for music, assuming 4 octaves or less. Finally, the results depend on the number of wavelet coefficients, as well as the set threshold. Generally, more filter taps mean greater reduction in operations, but lesser PSNR readings. Higher thresholds imply

greater reduction in operations, but their impact on PSNR values is less clear.

Sheu, et al. also eliminated multipliers from their architecture [Sheu96]. Instead, they use a look-up table, which saves on area. This has the obvious disadvantage of low precision. Also, changing the wavelet coefficients would be difficult if not impossible.

Lewis and Knowles present a VLSI architecture that performs the 2-D DWT with 4 tap Daubechies filters [Lewis91]. They take advantage of the wavelet coefficients and the low precision requirements of video applications to eliminate multipliers, which saves on space while increasing speed [Lewis91]. The four values used in this wavelet transform are:

$a = (1 + \sqrt{3})/8$
$b = (3 + \sqrt{3})/8$
$c = (3 - \sqrt{3})/8$
$d = (-1 + \sqrt{3})/8$

low pass filter:
$h = (a, b, c, -d)$
high pass filter:
$g(n) = (-1)^{n+1}h(3-n)$

Lewis and Knowles note that each data value must be multiplied by a, b, c, and d during the transform. With floating point arithmetic, a left shift of 1 bit accomplishes multiplication by 2. For example, multiplying by 3 can be easily accomplished by adding a data value to a left shifted version of the data value. Shifting a data value twice and adding the original value to it effectively multiplies it by 5. When performing image processing tasks, the wavelet coefficients round to a=11, b=19, c=5, d=3, since only 8 bits of precision are required. To get these nice whole values, calculate the above coefficients and multiply by 32, since multiplication by 32 is simply a left-shift of 5 bits. The latter 2 terms can be multiplied using a shifter and adder for each, as demonstrated above. The last two wavelet coefficients times the data value can be used to compute the data value multiplied by the first two. Let v represent the original data value, and 16v represent v left-shifted 4 times. To generate 11 times v, subtract 5v from 16v. To generate 19 times v, add 3v to 16v. In this way, four shifters

and four adders can multiply a data value by all four wavelet coefficients. The resulting convolver is only 1/8th the size of a normal convolver. While Lewis and Knowles only talk about the 1-D filter, the multiplier-less convolver naturally extends to the 2-D case. Line delays replace the delay elements before feeding the results back to the input multiplexors. Although this creates a larger area requirement, it cuts the memory bandwidth in half.

These algorithmic variations can be implemented according to the application's demands. For the case where all octave outputs are needed at once, Aditya's algorithm and associated architecture would be best. Chang's approach gives quick, approximate solutions. For fast, low-resolution video applications, Lewis and Knowles' multiplier-less algorithm would work well.

3.6 CONCLUSIONS

Architectures for the DWT share some common traits, since most designs have converged to a set of standards. First, finite impulse response filters are used. Most designs are scalable, so switching to a different wavelet is not a concern. Digital designs are the most common. Folding is the dominant design choice, since it is flexible and allows an architecture to do more without adding much area. In other words, doubling the amount of calculations does not mean doubling the area needed. Though Mallat's pyramid algorithm is the basic algorithm used, Fridman's work optimizes the algorithm with scheduling for folded architectures.

Options include the type of filter (serial versus parallel). Also, time versus space mapping allows the designer the trade off between area and latency. The target application will influence the architecture's design to an extent. For example, when speed is a critical factor, space mapping will give the best speed performance.

4. PROPOSED 3-D DWT ARCHITECTURES

4.1 THE NEED FOR A 3-D DWT

As was shown in Chapter 1, medical data needs a true 3-D transform for compression and transmission. Sample images from one MRI data-set (figure 4.1) show that any two consecutive images are similar, but naturally progress from front to back. This is much different from other image sequences, such as the "Miss America" sequence where the person talking moves a bit, but any two images in the sequence could be identical. Note that the images shown in the MRI figure are actually reversed to make them more pleasing to look at in print.

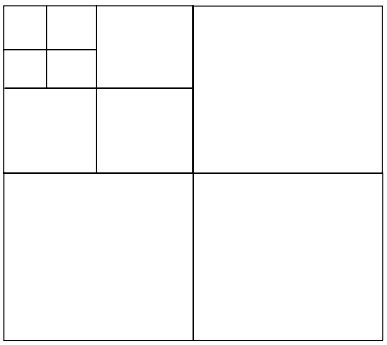
Figure 4.2 shows the decomposition of 1, 2, and 3 dimensional data. It assumes that each signal is broken down into 3 levels of resolution. The 1-D case shows that a 1-D signal of 32 values would be broken down into 16 first octave details, followed by 8 second octave details, followed by 4 details of the third octave, and finally 4 signal approximation values also generated in the third octave. The 2-D decomposition has 3 detail blocks in the first octave. These appear as the 3 largest blocks. The 3 blocks of the second octave are each one quarter the size of the remaining large block. The four smallest blocks represent the 3 details of the third octave, and the third octave approximation signal. The 3-D graphic shows how the first octave detail signals comprise 7/8 of the transformed data size. The second octave again breaks up the remaining data into 7 detail blocks and 1 approximation block. Finally, the third octave breaks the approximation block from the previous level into 7 details and 1 approximation. In summary, the data blocks are half the size for each level of resolution of 1-D data, one quarter the size for 2-D data, and one-eighth the size for 3-D data. For a realistic view of how rapidly the data gets

smaller for the 3-D case, a 256x256x32 MRI has 2Mbytes of data. After three levels of resolution, the approximate signal's size is only 4Kbytes.

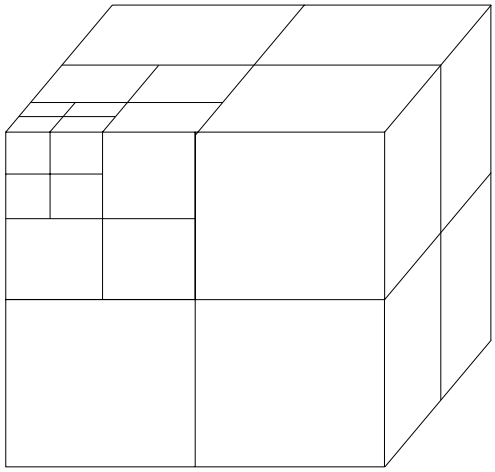
I am proposing 2 architectures for the 3-D discrete wavelet transform based on sequential and block design. The first architecture, called 3D-I, is a direct mapping of the 3-D DWT. It operates on data sequentially along row, column and slice (image). The second architecture, called 3D-II, operates on a block of data at a time. The architectures will be shown to have advantages and disadvantages, so both designs are useful.

Figure 4.1 - Sample MRI images

1-D, 3-octave decomposition



2-D, 3-octave decomposition



3-D, 3-octave decomposition

Figure 4.2 - Decomposition of data

4.2 3-D DWT ARCHITECTURES

The 3-D wavelet transform is shown conceptually in figure 1.4. Implemented directly, it would take 14 filters, as shown in figure 4.3. However, the filters 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. This can be seen in figure 4.4, where input data flows from left to right, undergoing the DWT in the X, Y and Z dimensions. In this figure, the original data size is 4x4x4. The shaded blocks show the how the downsampling affects the

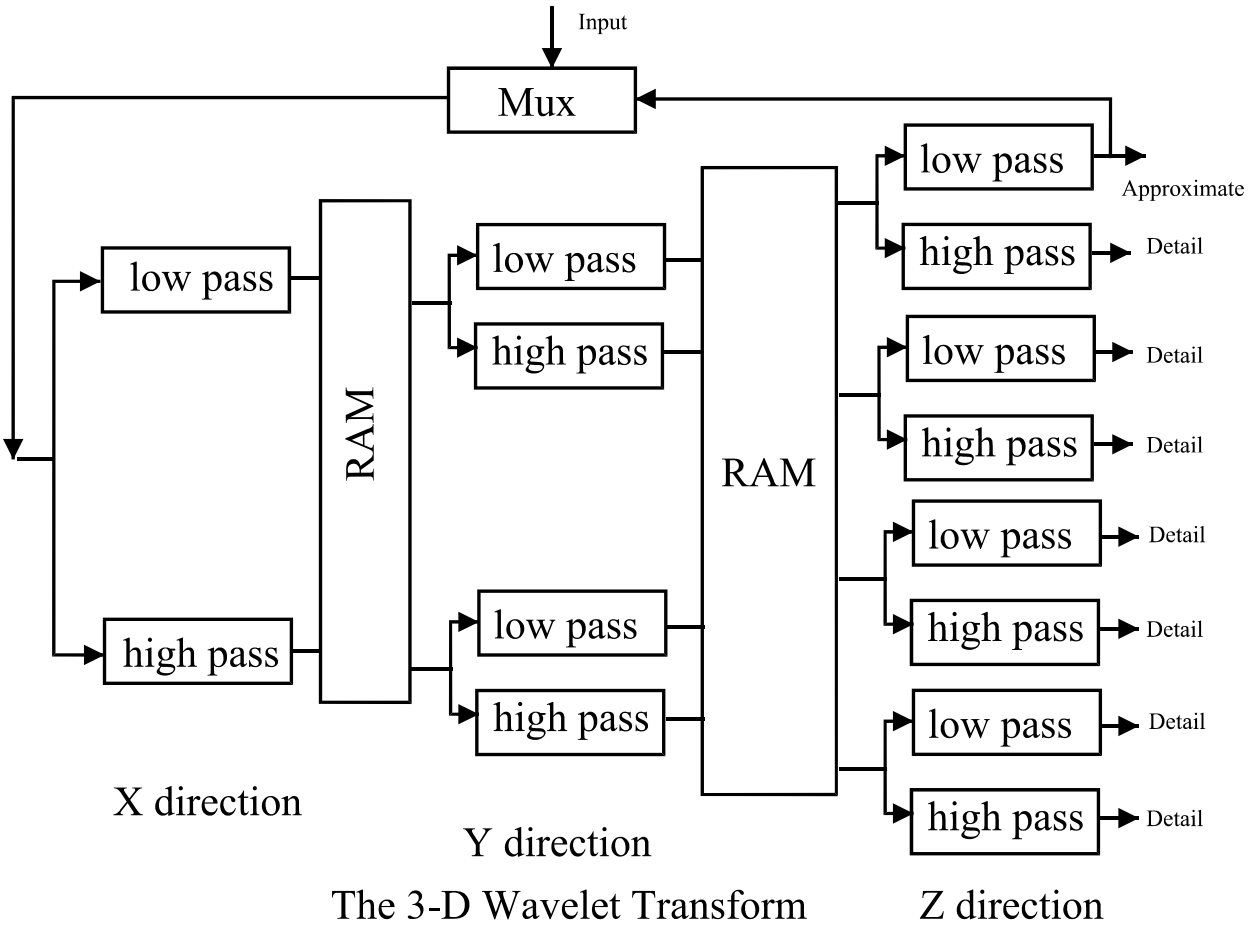


Figure 4.3 - The Conceptual 3-D DWT Design

data as it passes through the transformer. One way to balance the amount of work among filters is to fold the design, much like Parhi did with the multiresolution data for the one-dimensional case. For this design, the data is pipelined such that one filter pair calculates all outputs for one dimension.

Figure 4.5 shows how the conceptual model of the 3-D might be implemented directly, assuming the filters are folded. This design is the first architecture, called 3D-I.

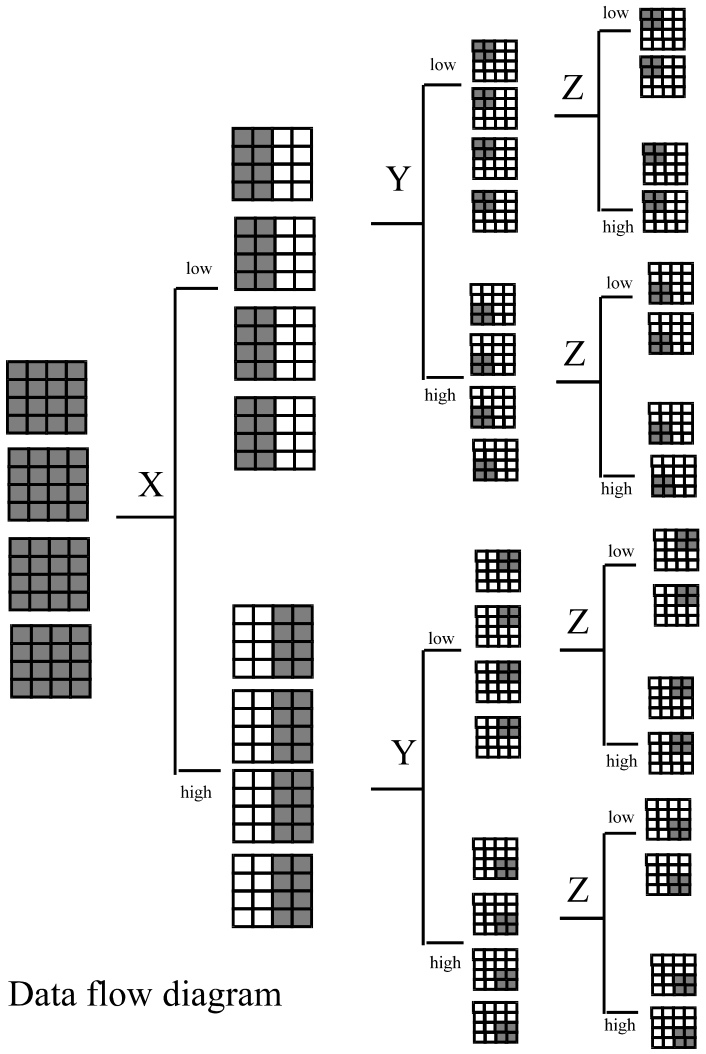


Figure 4.4 - Data flow through the 3-D DWT

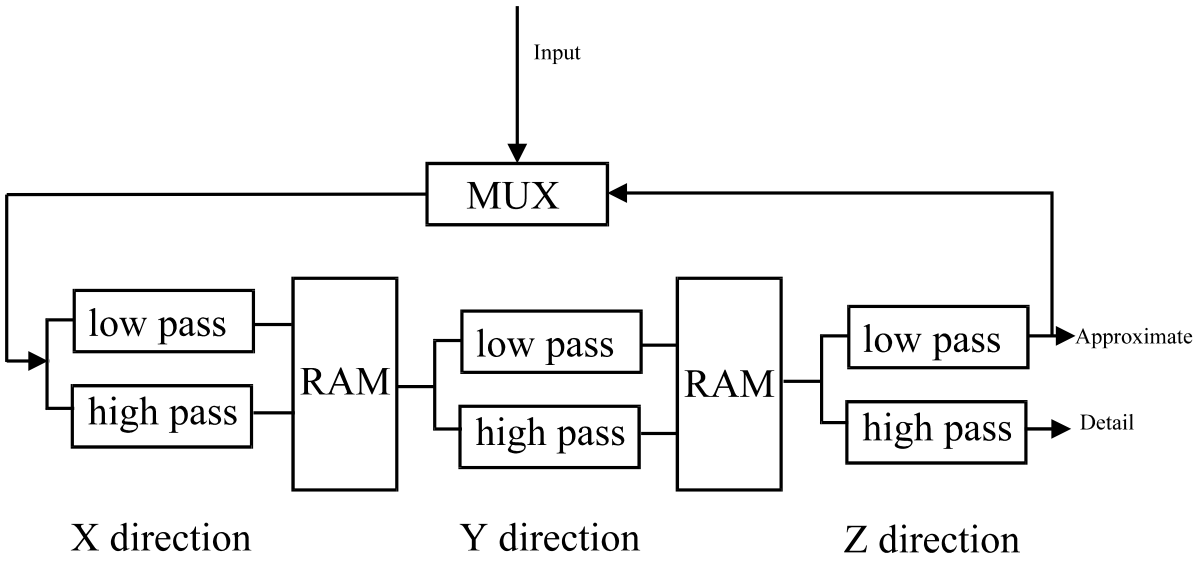


Figure 4.5 - The 3D-I Architecture

4.3 THE 3D-I ARCHITECTURE

The 3D-I architecture uses systolic filters, each containing L_i Multiply-Accumulate Cells (MACs). Here, L_i stands for the maximum number of filter taps in the dimension that the filter is processing. The MAC shown in figure 4.6(a) would be fine for the X dimension. For the Y and Z dimensions, the latches must be replaced with shift registers as shown in figure 4.6(b). The number of shift registers per MAC is dependent upon the dimension only, not the input size nor the filter size. The MACs are strung together as shown in figure 4.6(c). 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. Another way to move the data to get inherent downsampling is to send it from the even PE's to other even PE's, while the odd PE's send data to the next odd PE. This is what Fridman and Manolakos did in their design of a 1-D filter [Fridman94a]. Systolic filters allow scalability to higher filter sizes. A systolic filter can be expanded for 2 more coefficients easily by adding 2 more MACs in sequence. Expanding a parallel filter for 2 more coefficients is not so easy, since another level of

adders will need to be included.

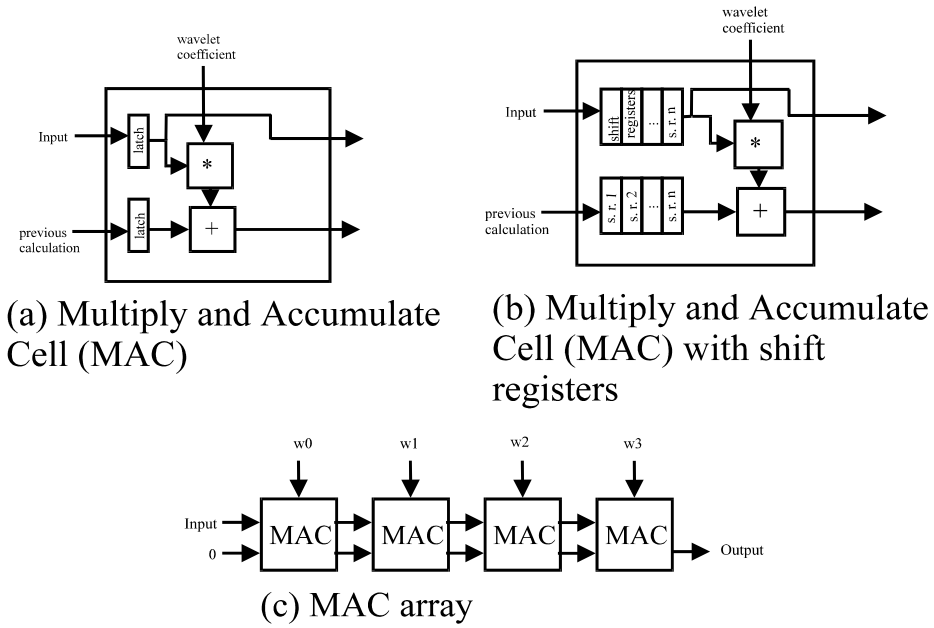


Figure 4.6 - The serial filter constructed from MACs

The design of the low-pass filter with a simple systolic array has the PE's passing data 25% of the time, and doing useful computations another 25% of the time [Syed95]. Having one filter do both low and high pass computations increases the hardware utilization. The PE's would hold onto the input values for an additional cycle. The PE's would need to store 2 wavelet coefficients, one for the low pass and one for the high pass. A control bit would determine which wavelet coefficient to use. The PE's of the simple 3-D DWT do either low or high pass computations. For the X direction, the PE's appear as in figure 4.6(a). However, for the Y direction, the PE's are equipped with extra registers and additional control to facilitate 2 low pass or 2 high pass computations on the same filter. In the Z direction, this is also true, only the filter will do the work of 4 low pass or 4 high pass

filters. For each dimension, the number of computations stays the same. The number of filters implemented doubles due to branching, but the amount of data passing through the filters is cut in half by the decimation operation. The decimation operation is not explicitly done, since the filter does not compute values only to discard them.

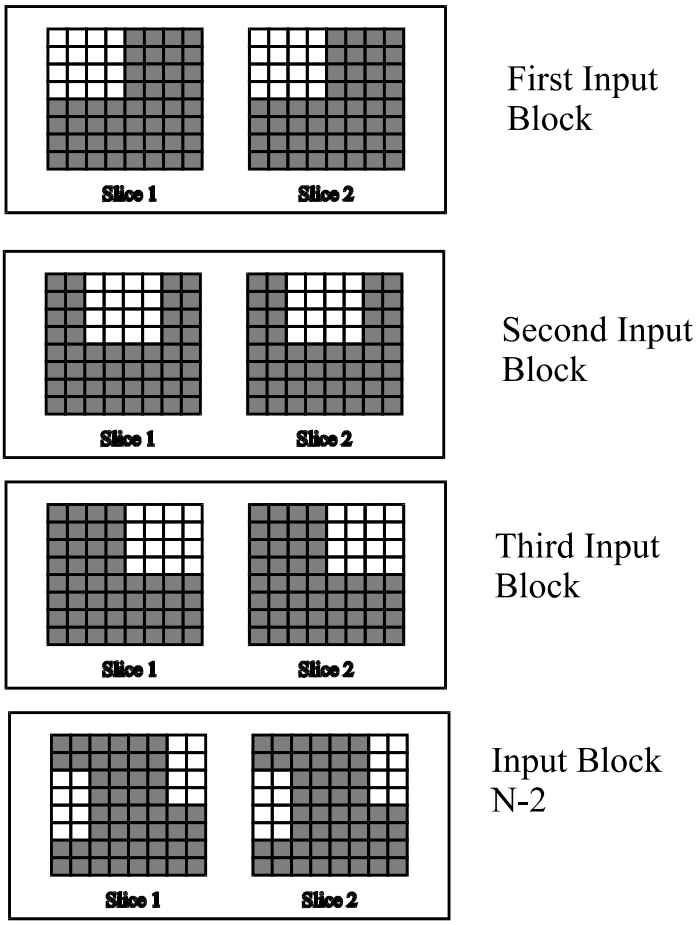
The 3D-I architecture has a folded design to allow scalability to a longer wavelet. The wavelet sizes determine the number of PEs in each filter. It has simple, distributed control, since each processing element has few functions: shift inputs, multiply and add. The PEs have only a few internal registers, which are consistent through the filter. In other words, any two PEs in the X filter have the same amount of registers. The 3D-I is cascadable, like other folded designs. Finally, the serial filters generate results in a low number of clock cycles.

In the 3D-I architecture, there are several drawbacks. To get started in the Y direction, the X direction must generate enough results, one row of outputs for every wavelet 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, then the Y filters followed by the Z filters. This analysis is expanded upon in the next chapter.

4.4 THE 3D-II ARCHITECTURE

The second architecture does not require large on-chip memory. The 3D-II reads the data

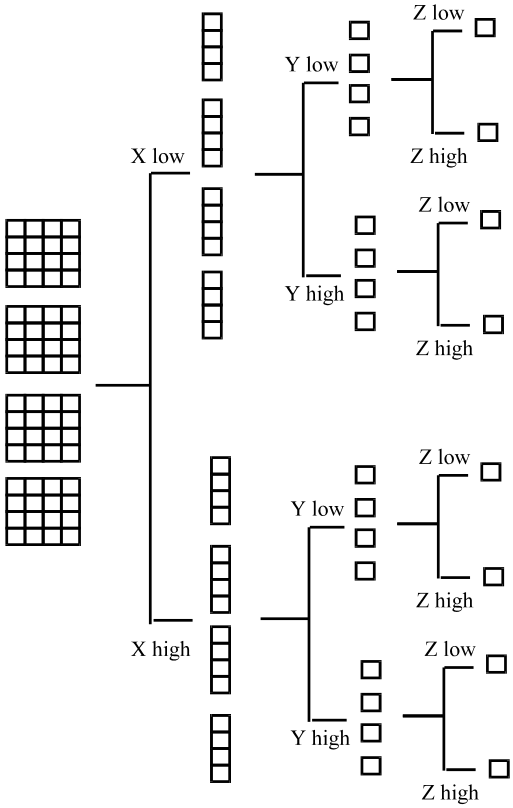
as blocks instead of the row-column fashion. Of course, a large RAM could buffer the row-column data and send it out in block format. With this capability, the 3D-II processor needs a data block of size L1*L2*L3 to compute the 8 output streams of the 3-D DWT. Figure 4.7 shows what this looks like for the filter sizes 4, 4, and 2. If another wavelet were used instead, the only changes besides the filter lengths would be the block size and on-chip memory. The block size would have to be 9x9x2 for the 9-7 biorthogonal spline, since the 9 and 7 sizes apply to the low and high pass filters respectively, and the data in each dimension will be sent to both low and high pass filters. Figure 4.7 shows how the transformer would request the data blocks. The block selector moves from left to right and front to back. The blocks are read, skipping every other block horizontally, vertically, and between images in order to take downsampling into account. The blocks could come in a different pattern, however the output from the 3D-II architecture would then be in a different order than in the 3D-I.



How the data blocks are read.
Since filter lengths are 4, 4, and 2, block size is 32.

Figure 4.7 - Block reads for the second architecture

Figure 4.8 shows how one block of data becomes 8 outputs for the 3-D DWT for a 4x4x4 transform. First, a 4x4x4 data block goes through the X filters. Each X filter takes 4 inputs and produces 1 output. So the intermediate data is now 2 blocks of 1x4x4. Next the Y filters each take 4 inputs and produce 1 output, forming 4 blocks of size 1x1x4. Finally, the Z filters take in 4 inputs and produce 1 output each. Thus, the end result is 8 blocks of 1x1x1, or simply 8 outputs. This holds for any size filter - a $L_1xL_2xL_3$ data block will become 2 $1xL_2xL_3$ blocks, then 4 $1x1xL_3$ blocks, then 8 1x1x1 blocks as it passes through the transformer.



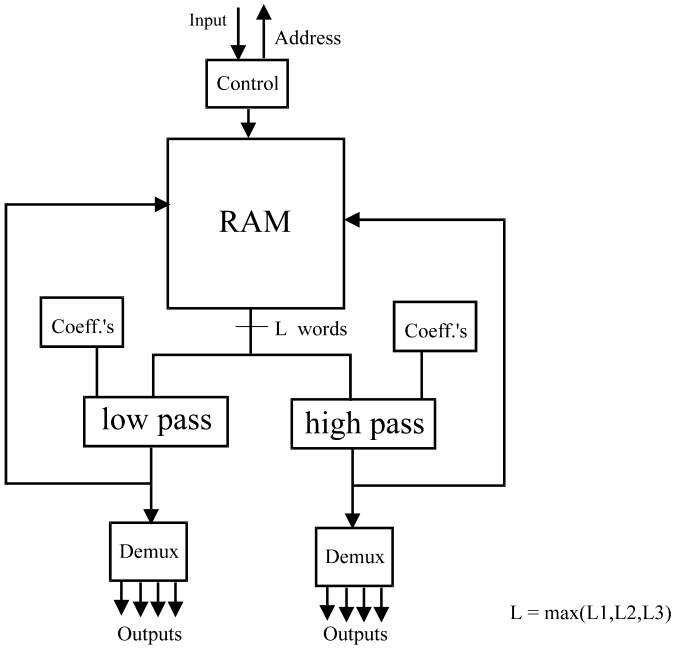
How a data block becomes 8 outputs for the 3-D DWT with filter sizes of 4, 4, and 4.

Figure 4.8 - How the data block becomes 8 outputs

The resulting architecture is shown in figure 4.9. The centralized control unit will be

more complex for this architecture than the prestored control of the previous one. First, it will be directly responsible for selecting the input block from off-chip. The off-chip memory could even be a hierarchy, such as a RAM and cache setup. The control unit generates the addresses needed to get the correct input block. Though the figure does not explicitly show it, the control unit will also select which coefficients to pass on to the low and high pass filters. The memory on the chip will be small compared to the input size. Inspecting figure 4.8 shows the amount of storage needed. It depends solely on the filter sizes, that is, it will be $L_1xL_2xL_3 + 2xL_2xL_3/L_1 + 2x2xL_3/L_1xL_2 + 8$. The last 8 values do not need to be stored on chip, but are outputs. The demultiplexor units select which of the outputs are being generated during the last four clock cycles. The design needs $1*L_2*L_3$ cycles to do the X calculations, followed by $2*L_3$ cycles to do the Y calculations, followed by 4*1 cycles for the Z calculations. Note that the last 4 cycles produce 2 outputs each. This results in $(L_2 + 2)*L_3 + 4$ cycles per every 8 outputs.

Multiplexing the calculations in space instead of time is the best choice for this architecture. This design choice is made due to the fact that time multiplexing slows down the architecture. For the 3-D DWT, speed is very important. There is one low and one high pass filter calculation done in every cycle. Also, the filters should be parallel, since this produces the computation with the least latency. Systolic filters assume that the data is fed in a non-block form such that partial calculations are done. In each clock cycle of the 3D-I, a systolic filter does 1/L of any calculation. It works on L calculations at the same time, but only completes 1/L of any one calculation. The systolic filter is very efficient, but assumes partial calculations that result in large memory requirements. In the 3D-II, calculations are done on one block as an atomic operation. No partial calculations are done, and a minimal latency is needed.



The 3-D Wavelet Transform with block inputs and parallel filters.

Figure 4.9 - The 3D-II Architecture

The coefficients unit is shown in figure 4.10. This unit acts as a set of 3 registers with a multiplexor. Registers 0,1, and 2 contain the wavelet coefficients corresponding to L_1, L_2, and L_3, respectively. The registers are shown with two inputs, a one-bit line specifying load, and a bus with the coefficients for initial loading. The width of the bus is max(L_1, L_2, L_3)*precision. Since the registers are loaded from off-chip, the coefficients are programmable. The multiplexor selects which coefficients to pass on to the filter unit based on the counter. The bus width is 64 bits, figure 4.10, which corresponds to a 4x4x2 wavelet transform.

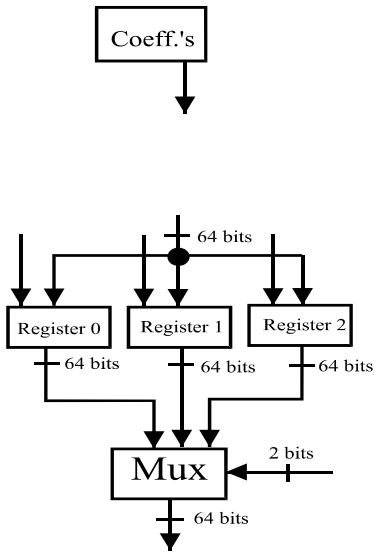


Figure 4.10 - Details of the Coefficients Unit

The RAM unit, figure 4.11, stores the input blocks and the results from the filters after X and Y dimension calculations (noted low results and high results). Each clock cycle, the RAM unit will send out max(L_1, L_2, L_3) words for the next calculation. Also, it will store the 2 results from the previous calculation. The control signal specifies the write locations. The input data is max(L_1, L_2, L_3) words, which are sent from off-chip. The output data always has max(L_1, L_2, L_3) words, one word per wavelet coefficient. The figure shows the line widths for a 4x4x2 wavelet transform and 16 bits of precision. Note that the RAM needs at most 64 words in this implementation, but as shown in the next chapter, only 56 are actually necessary. The last 8 locations would store the outputs, but these are sent off-chip.

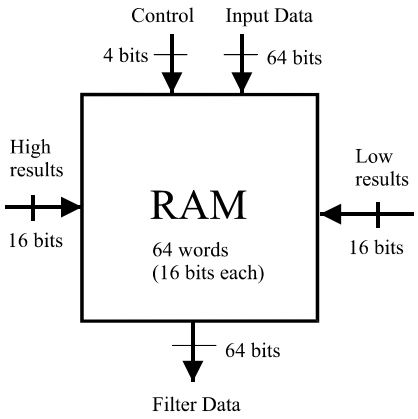


Figure 4.11 - The RAM Unit

The heart of the wavelet processor is the filter. The 3D-II filter unit, figure 4.12, shows how the multipliers connect to the adder tree. The two units labeled "low pass" and "high pass" are identical. They are two instances of the filter unit, with the coefficients differentiating them. The example unit shown multiplies the max(L_1, L_2, L_3) (4) coefficient inputs with the max(L_1, L_2, L_3) (4) data inputs. The input busses allow max(L_1, L_2, L_3) (4) words to travel in parallel, but they are split apart and routed to the appropriate multiplier inside the filter. The adder tree adds the results together, which are then sent to either the RAM unit, or off-chip. For an example input of w0, w1, w2, w3 wavelet coefficients and data of a, b, c, d, the result produced by this unit is a*w0 + b*w1 + c*w2 + d*w3. The multipliers keep only a certain number of the output bits, since a 16x16 bit multiplication produces a 32 bit result, so the results are trimmed back to the amount of precision. The next chapter goes into more detail about this process.

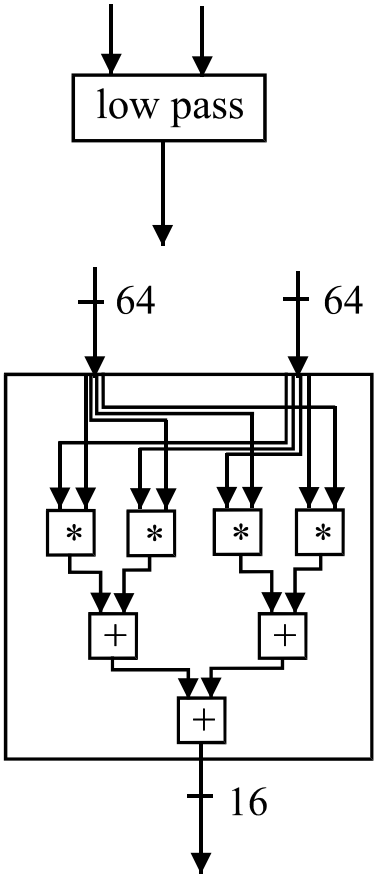


Figure 4.12 - The Details of the Filter Unit

The final detail view of the 3D-II architecture is the control unit, figure 4.13. This unit provides the control signal, which is used for many things, such as the RAM read and write locations, the wavelet coefficient to use, and when to send the results off-chip through the demultiplexor. In the example implementation, the control signal is a 4 bit counter. It has a clock input and a data input, though the data passes through to the RAM unit. Besides the control signal, the control unit generates the data block's address for the off-chip memory. Obtaining the whole block from one address would make this architecture simpler to design. However, this capability would require a more intelligent off-chip memory system than just a simple RAM

configuration. Therefore, it is assumed that one block request must be broken down to L_2*L_3 requests of L_1 values. If the input data is stored as 3-D data in a large RAM, then the following equation generates the addresses needed:

$$\text{Address} = (\text{frame} + j)*\text{frame\_size} + i*N + \text{column\_offset} + \text{row\_offset}$$

The control unit therefore needs to know a bit about the data dimensions. N, M, and P specify these, representing height, width, and number of images, respectively. Index variables i and j allow the address generator to get a block based on the filter sizes. The chip will read word size of L_1 data values at one time. Thus, the chip will need to get L_2*L_3 words, which defines the block size. Index i goes from 1 to L_2 while index j counts from 1 to L_3. Multiplications are not really necessary, if a large adder updates the variables instead of using indices. To simplify the design, the following equation is used:

$$\text{Address} = \text{frame} + \text{row} + \text{column}$$

This equation assumes that frame counts by the size of the frame (M*N), and row counts by the row size (M). Upon reset, frame = frame_offset, row = row_offset, and column = col_offset. After reading a word, the row register adds M*2 (the M register is wired such that it appears as 2M to the row adder). After reading L_2 words, the frame register adds frame_size to itself, and another L_2 words will be read. This process repeats L_3 times. After reading a complete block, the contents of the offset registers are updated to point to a new block. The col_offset register is updated by a value of 2, its new value is compared to M, and column_offset is reset if it is found

to be greater. Resetting the column register triggers the row_offset register to switch to a new row, which is compared to the maximum in turn. Similarly, resetting the row_offset will update the frame_offset register. When the frame_offset register triggers a reset, the transform is complete. Figure 4.13 only shows the comparator and adder circuitry with the column variables for greater clarity. The row and frame registers have a structure identical to the column's comparator and adder circuitry, except for the inputs.

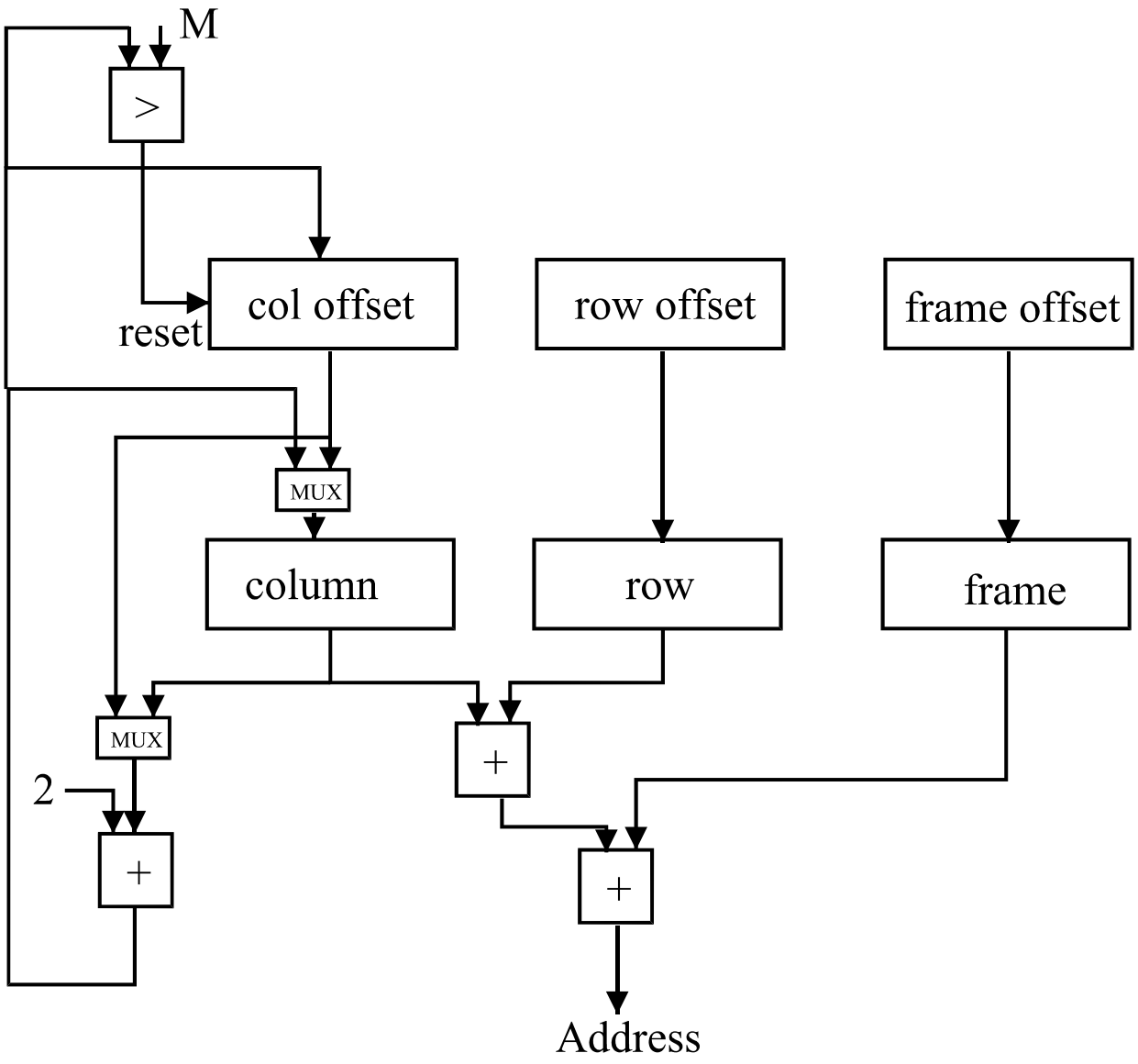


Figure 4.13 - The Address Generation Details of the Control Unit

5. ARCHITECTURE EVALUATION

5.1 WAVELET COEFFICIENTS

The wavelet coefficients are programmable in both designs. Programmable simply means that the coefficients are loaded from off-chip. Therefore, a user could have the chip perform the DWT with one type of wavelet, then perform it again with another wavelet. Note that programmable is not necessarily the same thing as scalable, since scalable implies that an arbitrarily large wavelet can be used. Typical wavelet coefficients appear in Appendix A. In the 3D-II design for 4x4x2, the first and second registers of the filter units will contain Daubechies coefficients. The third register will contain Haar coefficients. The reconstruction filter coefficients are included as well. These are not used in our implementation, but would be used in a slightly modified version of the 3D-I architecture to reconstruct the original signal. Basically, doing the reconstruction requires another adder and timing adjustments.

5.2 FIXED POINT versus FLOATING POINT

Another question is whether to use fixed-point multiplications, or floating point? Fixed point has the advantages of being easier to implement, requires less silicon area, and makes multiplications faster to perform. Floating point allows a greater range of numbers, though floating point numbers require 32 or 64 bits. Fixed-point numbers can be 8 to 32 bits (or more), possibly saving space in the multiplier.

According to [Chang97], "Because of the compact support and the small magnitudes of coefficients, the WT using these types of coefficients does not fully utilize the wide dynamic

range provided by the floating point numbers". Also, "Computing WT using fixed point numbers reduces the hardware complexity". In other words, the extra hardware that floating point requires is wasted on the DWT. Chang's design uses the Booth multiplier, a standard design for integer multiplications. Another researcher agrees, "In order to meet the real time requirements of applications such as video compression, a fast multiplier design is required. For this purpose, a high-speed Booth multiplier is used in the filter cell" [Grzeszczak96].

Limqueco implemented his design with fixed-point computations [Limqueco96a]. In the horizontal and vertical filters, decimal numbers including some bits for the fractional part represent the values. When the results leave the filters, the values are rounded and truncated to an 8-bit format.

Grzeszczak also uses fixed-point numbers. "The dynamic range of the DWT coefficients will increase because of the sum of several terms" [Grzeszczak96]. The increase is upper-bounded by approximately 2. An extra bit of precision for each transform octave or dimension will be needed, thus the 3-D DWT needs up to 3 extra bits internally for one octave. For a 1-D architecture, Grzeszczak notes "the simulation results in section III shows that it is sufficient to have an architecture with 12b precision" [Grzeszczak96]. Other designers also use fixed point, such as Lewis and Knowles, who use 8 bit fixed point arithmetic for data and coefficients. Results were rounded back to 8 bits for the next stage of computation. Wang and Huang say that wavelet transformed data is naturally floating point, but they round it to the nearest integer values, to minimize data loss [Wang95]. Aware's chip has 16x16 size multipliers, but keep 16 bits of the result, which is chosen by software. It has 16 bit adders, and its I/O data size is 16 bits [Aware94]. Therefore, fixed-point numbers with an assumed radix are used in both proposed architectures. Since the Aware design was fabricated, this design choice has been proven

successful.

5.3 PRECISION

The input grey-scale values are 8 bits wide. The wavelet coefficients can be stored as 8 bits or 16 bits (or more), depending on the designer. The multiplication result has 16 bits (assuming an 8 bit x 8 bit multiplication). The question is how much precision to keep? Should the internal registers be 16 bits wide, or are 8 bits enough? Also, how much precision should the output have?

The output should have the same precision as the input. After all, the purpose of the transform is to ultimately reduce the amount of space needed by the data. It doesn't make sense to transform N pixels of 8 bits each into N data samples of 16 bits each. So in the end, inputs are 8 bits wide, intermediate values are 16 bits, and outputs rounded back to 8 bits. The one exception is with the approximate signal, which stores the outputs as whole integers, even though the values may exceed 255.

The only way to keep the intermediate data at 8 bits would be to assume that the low pass outputs are always positive, and range from 0..255. Also assume that the high pass outputs range from -128..127. This works for this particular (Haar) wavelet, but cannot be generalized for other wavelets. The Daubechies 4-tap wavelet, for example, has a negative as one of the low pass filter coefficients. This means that the low pass filter output could be negative. Since the wavelet transform architectures have programmable wavelets, the chip would have to use truncation on the filter outputs to keep the results at 8 bits. This is impossible with the Daubechies wavelets. After performing the 3-D DWT on a 256x256x32 grey-scale MRI data set with 4x4x2 wavelets,

the minimum value was found to be -245, while the maximum value was 611. The results would need a wider range than 8 bits provides, therefore the 3D-II implementation uses 16 bits between filters. Eight bits just is not enough space; packing the data into 8 bits loses some precision. When precision is lost, an exact reconstruction is not guaranteed. Even with the Haar coefficients (1/2, 1/2, and 1/2, -1/2), there must be one extra bit for the sign. For example, assume the following data stream of 0, 255, 255:

Low pass outputs:	High pass outputs:
(0+255)/2 = 127	(0-255)/2 = -127
(255+255)/2 = 255	(255-255)/2 = 0

The resulting streams have a range of 0..255 (needing 8 bits for this), as well as negative numbers (which requires 1 extra bit). Therefore, even for the Haar coefficients, the width of the intermediate data "grows" by 1 bit. If truncation were used instead, keeping the lower 7 bits and using the 8th bit as the sign, the reconstruction would not be able to distinguish a "0" from "128", since 0 AND 127 = 0, and 128 AND 127 = 0.

Sixteen-bit coefficients were chosen to be a good size for intermediate values (i.e. first octave outputs). Most papers that address this use either 8 or 16 bits. For example, Aware's processor uses 16x16 multiplier, but only keeps 16 bits of the output. One author (Grzeszczak) does some analysis on the word size, and shows that 32 bits is too much.

5.4 LOWER OCTAVE SCHEDULING

The first architecture, 3D-I, can handle lower octave computations by scheduling them

between the first octave calculations, such as the work of Fridman and Manolakos. The second architecture operates on only one octave at a time. To have it do additional octaves, the low-low-low pass data can be fed in as a next data set. Scheduling the 3D-II architecture to perform lower octave computations is troublesome, because of the following difference. The first architecture performs calculations with sequential filters, a series semi-systolic processing elements, which lend themselves to partial calculations. The second architecture does all calculations for 8 outputs as an atomic calculation. Storing partial results for later use requires additional memory and an increase in control complexity. The strong point of the 3D-II architecture is its small on-chip memory, which would be defeated if it had to store partial results for later use.

5.5 3D-I MEMORY REQUIREMENTS

The outputs of the previous filters and the requirements of the next filters determine the amount of memory between filters of the 3D-I architecture. For example, between the X filters and the Y filters, there must be enough memory to store the row-wise outputs. The Y filters, however, read their data in a column-wise fashion. Thus, they must wait until one complete image worth of data is available so that they can synchronize with the X filters. Having the Y filters start early means that they will try to read a location before a valid X output is stored there, figure 5.1. Having the Y filters start later means that X outputs will over-write a location before the Y filters have had a chance to read it. To synchronize the X and Y filters, there should be enough memory between the two filters that equal the size of one image. Considering folding of the Y filters, the data flow works as follows. First, the X filters fill up the low pass memory and high pass memory. The X filters put a value in one of these sections each clock cycle, then Y

filters read two values from one of these sections at a time. It will get 2 values from the low pass memory on even clock cycles, then 2 values from the high pass memory on odd clock cycles. The reason Y filters get 2 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, while the other value goes to the first odd MAC.

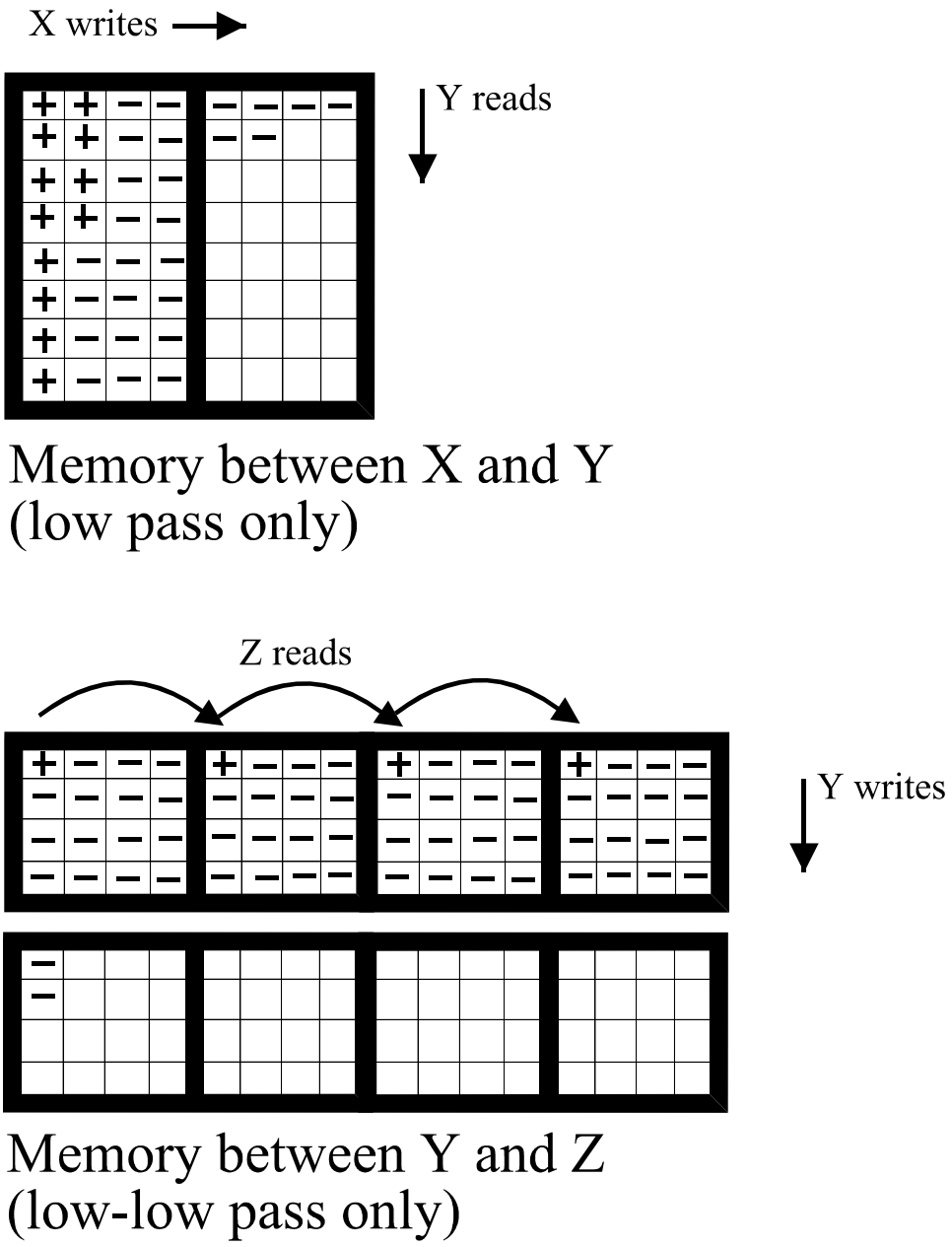


Figure 5.1 - The Memory Requirements for 3D-I

The Y inputs depend on the X outputs, while the Z inputs depend upon the Y outputs. Thus, the operations cannot be re-arranged. The X filters take their inputs in a column, row, slice fashion. The Y filters then take their inputs in a row, column, slice fashion. Then the Z filters go along the slices first. Suppose we label the height N, the width as M, and the number of images P. If X reads N*M*P inputs, then 2 (conceptual) Y filters will read N*M*P/2 inputs. Then 4 (conceptual) Z filters will read N*M*P/4 inputs. Each Y filter must wait until N*M/2 inputs are available, while each Z filter must wait until (N/2)*(M/2)*P of its inputs are available. The memory between the Y and Z filters is 4 sets of N*M*P/4, or N*M*P. Therefore, the design depends on the input data's dimensions. This is shown in figure 5.1 for a small input size of a 4 frame sequence of 8x8 images. A dash indicates that the memory location holds a valid input for the next filter set. A plus indicates that the memory location has been read by the next filter set, and is available for the next value. The arrows indicate the order that the data are read/written. Remember that each filter downsamples the outputs, and writes to two memories. This figure only shows the memory needed between the X low-pass filter and the Y filters, so the actual memory between X and Y is double the amount shown. Since the original data size is 8x8x4, the data size reaching one of the Y filter pairs is 8x4x4 due to the downsampling operation. Also 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 2*N*M, while the amount of memory required between Y and Z filters is (M/2)*(N/2)*P*2*4, or N*M*P*2. There must be 4 sets of the memory, one for each Y 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.

5.6 3D-II MEMORY REQUIREMENTS

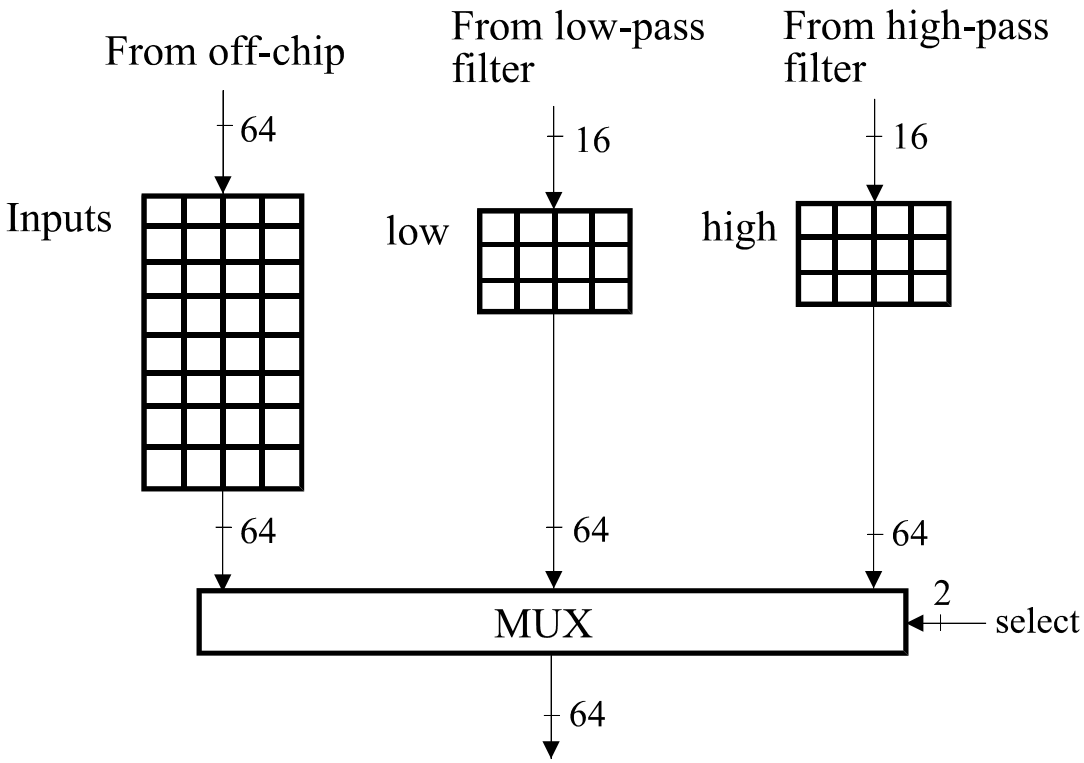
The RAM unit of the 3D-II architecture needs to store the input block, and the X and Y filter outputs. The RAM must conceptually read L_1 16-bit values each cycle, and write up to 3 values: L_1 16-bit input values (regarded as one large input), and 2 16-bit filter output values. Since allowing multiple write ports makes the layout more difficult and potentially larger, the author has chosen to modify this RAM to make implementation easier, figure 5.2. It can be broken down into three sections. The first section buffers the input from off-chip, and can be implemented as a group of shift-registers. It holds L_2*L_3 values, each one $16*L_1$ bits wide. Note that the squares representing memory locations are each 16 bits wide. Also, the figure assumes a wavelet size of 4x4x2. Thus, the shift register block will have a $16*L_1$ wide input (64 bits here), a $16*L_1$ wide output (64 bits), and a "store" input signal that indicates when to store the input busses' contents.

The second and third sections combined must store a total of $(L_2*L_3*2 + L_3*4)$ locations (24 in this example) of 16-bit registers. This combined with the input block storage gives the total amount of memory needed below:

$$\text{Total memory needed by 3D-II} = (L_1*L_2*L_3 + L_2*L_3*2 + L_3*4) * \text{Precision}$$

Though the memory writes 16 bits at a time, it reads $\max(L_1, L_2, L_3)$ 16-bit values (here 64-bits) at once. Section 2 holds the low pass filter's outputs, while section 3 holds the high pass filter's outputs. Each section of memory outputs $4*L_1$ bits at a time, which can pass through a

multiplexor. The select lines for the multiplexor are simply the 2 most significant bits of the global counter. On a 00 or 01, it should pass on values from the shift-registers. A value of 10 selects the low pass memory, while a value of 11 selects the high pass memory. Similarly, the control signal can be used to select which memory section to read. Also, the control signal tells the low pass memory and high pass memory where to store the intermediate data. These two memory sections have 12 places to store data, and these are accessed sequentially governed by the counter. A counter value of 1100 or higher means that no memory writes need to be done for the moment.



Modified RAM Unit

Figure 5.2 - The Modified RAM Unit of 3D-II (Shown for a 4x4x2 Wavelet Transform)

5.7 SERIAL versus PARALLEL OPERATION

The 3D-I architecture, since it performs all the X calculations in a sequential order, will produce partial calculations on data in a semi-systolic array. Thus it would appear that the 3D-I architecture has equal performance as the 3D-II, if not better. However, the 3D-II architecture can easily work in parallel with other 3D-II processors. The 3D-I architecture cannot. Placing 2 (or more) 3D-II processors next to each other does not even require revision of the architecture. All they would need is different (though slightly overlapping) data streams. The overlap comes at the boundaries. For example, if processor 1 does the 3D DWT on slices 1-16, and processor 2 handles slices 17-32, processor 1 will need to access slice 17 while processor 2 will need access to slice 1, assuming circular inputs and the Haar wavelet is used for the third dimension. If zero-padding is used instead, processor 2 will not need access to slice 1. If a longer wavelet (say it has R-taps) is used in the third dimension, then each processor would need R-1 slices from the other processor's data set, assuming circular inputs. Note that the data sets are not changed, so the memory serving the two processors can be separate RAM banks. Whether or not to use circular inputs can be left to the application. To use circular inputs, simply store the first R slices after the last slice. Since the 3D-II architecture can work in parallel with other 3D-II's, it will outperform the 3D-I architecture.

5.8 SIMULATION RESULTS

The results below in table 5.1 were obtained after performing the 3-D DWT on a 256x256x32 grey-scale MRI data set with the 3D-I simulation. It used Daubechies 8 coefficients for the X and Y dimensions, followed by the 2 Haar coefficients in the Z dimension.

The results in table 5.2 below were obtained after performing the 3-D DWT on a 256x256x32 grey-scale MRI data set with the 3D-II simulation. Wavelets used are Daubechies 4 in the X and Y dimensions, and Haar in the Z dimension. These results confirm the clock cycle analysis for the two architectures, which appears in the next 2 sections.

X dimension with Daub 8 coefficients
Start time = 0
Image height = 256
Image width = 256
Number of images = 32
Input width = 8
Xdir endtime = 1048581
Number of mults 16777312
Number of adds 16777312
Data read 2097152

Y dimension with Daub 8 coefficients (2 sets of RAM)
Start time = 32768
Image height = 256
Image width = 128
Number of images = 32
Input width = 16
Ydir endtime = 32768 + 1048584 =1081352
Number of mults 16777376
Number of adds 16777376
Data read 2097152 * 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
Zdir: endtime = 557056 + 1048592 = 1605648
Number of mults 4194384
Number of adds 4194384
Data read 2097152

Table 5.1 - The Simulation Results for the 3D-I Architecture

File: 'c:\mri\slices.dat' length = 2097152
 data dimensions: 256x256x32
 endtime = 4194304
 There were 262144 writes to disk (multiply by 8 values per write)
 Data count (words read) = 8388640
 Mult count = 33554432
 Add count = 29360128

Table 5.2 - The Simulation Results for the 3D-II Architecture

5.9 3D-I CLOCK CYCLE ANALYSIS

For the 3D-I architecture, there are $(L_1 + L_2 + L_3)*2$ multiplications per clock cycle, since there are L_1 multipliers per X filter, L_2 multipliers per Y filter, L_3 multipliers per Z filter, and each dimension has 2 filters. Similarly, there are $(L_1 + L_2 + L_3)*2$ additions per clock cycle. The number of clock cycles is determined by the input dimensions, as well as the lower octave scheduling. In order to compare these two architectures fairly, only single octave results are used. The number of clock cycles for the 3D-I design depends upon the input dimensions. The number of inputs, of course, is the height * width * depth, or N*M*P. Each filter pair needs N*M*P/2 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 + N*M/2 + N*M*P/2. The term $(L_1 + L_2 + L_3)/2$ 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 tiny compared to the overall time needed.

3D-I Total clock cycles = $(N/2)*(M/2)*P + N*M/2 + N*M*P/2 + (L_1 + L_2 + L_3)/2$

In the 3D-I, each MAC will do the following in one clock cycle:
t1 1 multiplication
t2 1 addition
t3 data shift and a partial calculation shift (done in parallel)

Thus the minimum clock cycle time for the 3D-I architecture is determined by: t1 + t2 + t3

5.10 3D-II CLOCK CYCLE ANALYSIS

In the 3D-II Architecture, One Clock Cycle Must:
Read the off-chip value:
x1 generate address (5 adds, 3 in parallel)
x2 send address off chip
x3 off-chip memory processes, returns word
x4 store word on-chip
Process the value:
y1 L_1 multiplications (in parallel) (assumed to equal t1 above)
y2 $\log_2(\max(L_1, L_2, L_3))$ parallel additions (assumed to equal t2*constant)
y3 write results back to on-chip memory

One way to do this is to schedule the operations as below (shown for L_1=4, L_2=4, L_3=2):

Step	Operation
1	Read the off-chip value 1
Process off-chip value 1	
…	
8	Read the off-chip value 8
Process off-chip value 8	
9	Read the on-chip value 1
Process on-chip value 1	
…	
16	Read the on-chip value 8
Process on-chip value 8; go to step 1	

The full schedules for the 4x4x2 and 4x4x4 cases appear in Appendix B. The advantage of the above schedule is that it uses less memory since the $L_1*L_2*L_3$*precision amount of values can be

reduced to L_1*precision. But this schedule has a dependency problem, where events must wait upon each other. The minimum clock cycle will be determined by:

x1 + x2 + x3 + x4 + y1 + y2 + y3 (nano)seconds

An alternate schedule is the following (shown for L_1=4, L_2=4, L_3=2):

Step	Operation
1	Process value 1; Read the NEXT off-chip value 1
2	Process value 2; Read the NEXT off-chip value 2
…	
8	Process value 8; Read the NEXT off-chip value 8
9	Process value 9 (from on-chip)
…	
16	Process value 16 (from on-chip); go to step 1

Though this schedule assumes the input data block will be stored in on-chip memory of size L_1*L_2*L_3*data_width, the clock cycle time will be reduced. The extra memory is a small price to pay. For example even with filter sizes of 10x10x2 and a data width of 16, this algorithm needs only 3200 - 160 = 3040 bits of additional storage. The minimum clock cycle will be determined by:

Max (x1 + x2 + x3 + x4, y1 + y2 + y3) seconds

The schedules also show how many operations are done on the data. For the 4x4x2 case, there are 16 clock cycles per block. Note that the number of clock cycles per block is not simply L_1*L_2*L_3. It is just a coincidence here, and will be explained in more detail below. The number of blocks per input size is constant, regardless of the filter sizes:

Number of blocks per input size = input size / 8

This number is intuitive, since the 3-D transform has 8 output streams, and each block input generates 1 values per stream. Thus the output size matches the input size.

The number of clock cycles needed per block varies with the filter sizes. It is assumed that the busses are L_1 values wide within the chip. The architecture processes the X dimension first, which will take L_2*L_3 clock cycles to read L_2*L_3 values and write L_2*L_3*2 values. The Y dimension will get L_2*L_3*2/L_2 values and store L_3*2*2 values. Finally, the Z dimension will obtain L_3*2*2/L_3 values and send 8 outputs off-chip. For every value read (or step), there must be 1 clock cycle, so that the total number of clock cycles per block = $L_2*L_3 + L_3*2 + 4$ clock cycles. Thus the total number of clock cycles for the 3D-II architecture is:

$$\text{3D-II Total clock cycles} = (N*M*P / 8) * (L_2*L_3 + L_3*2 + 4)$$

Each clock cycle of the 3D-II architecture involves $\max(L_1, L_2, L_3)*2$ multiplications, since there are two filters of size $\max(L_1, L_2, L_3)$. When the number of coefficients does not match the filter size, the coefficients are padded with zeroes. This contributes to the number of multiplications, though the multiplier's results are not used. The number of additions per clock cycle also depends on largest filter size, since it performs $\max(L_1, L_2, L_3)$-1 additions. The multiplications are done in parallel, while a tree of adders calculate additions, taking $\log_2(\max(L_1, L_2, L_3))$ amount of time.

The graphs below, figures 5.3 through 5.8, show how the two architectures compare with regards to number of clock cycles. The number of multiplications and number of additions directly depend on the clock cycles. These graphs show that the 3D-II architecture always needs more clock cycles than the 3D-I architecture. In many cases, the number of clock cycles is

hundreds of times less for the 3D-I architecture.

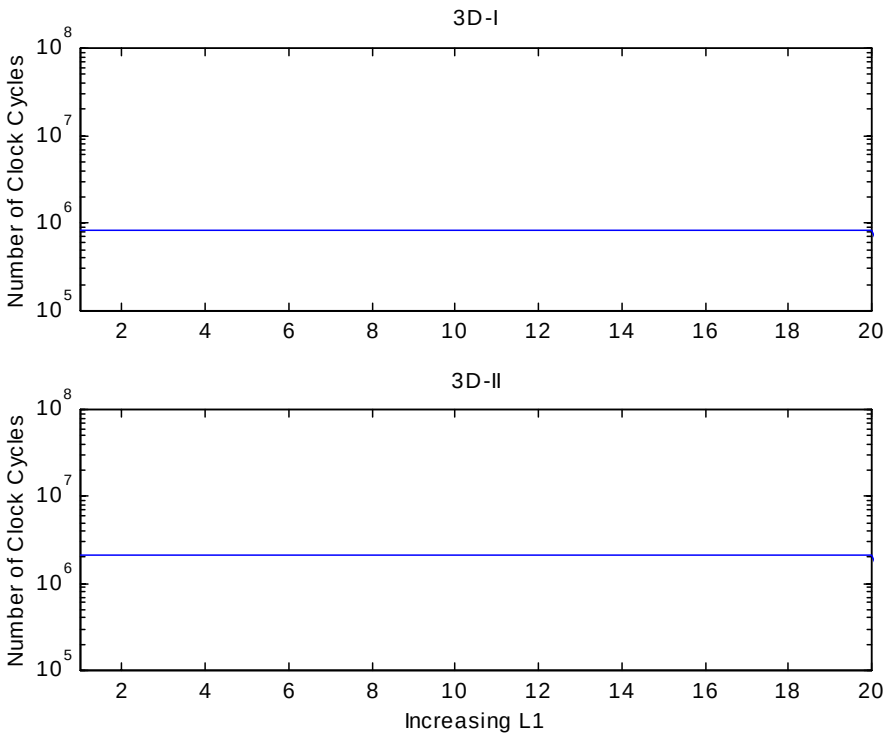


Figure 5.3 - Time Requirement (in Clock Cycles) for Filter Size L_1

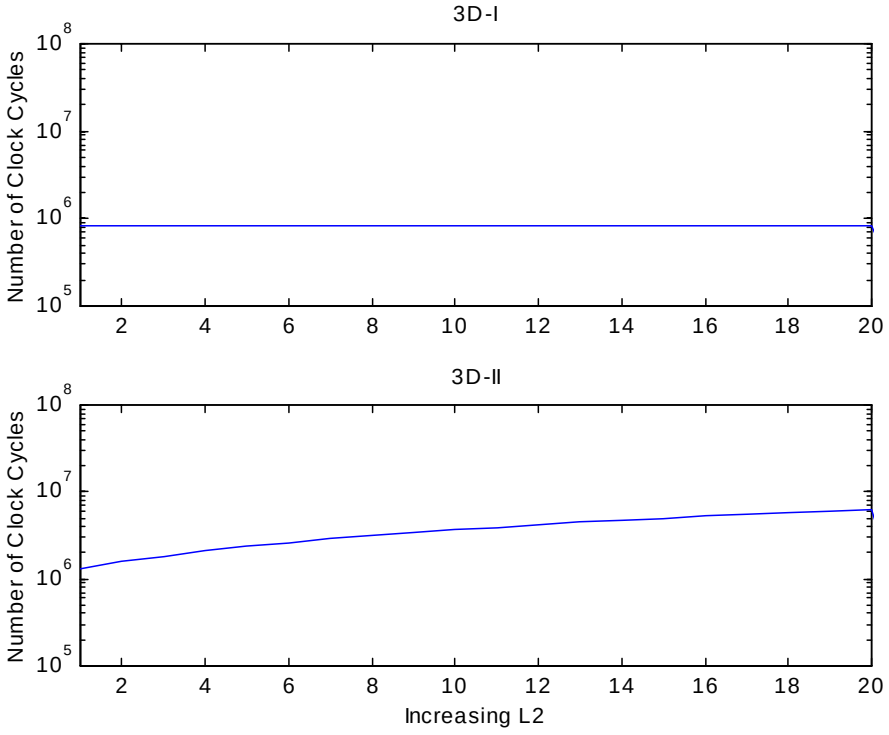


Figure 5.4 - Time Requirement (in Clock Cycles) for Filter Size L_2

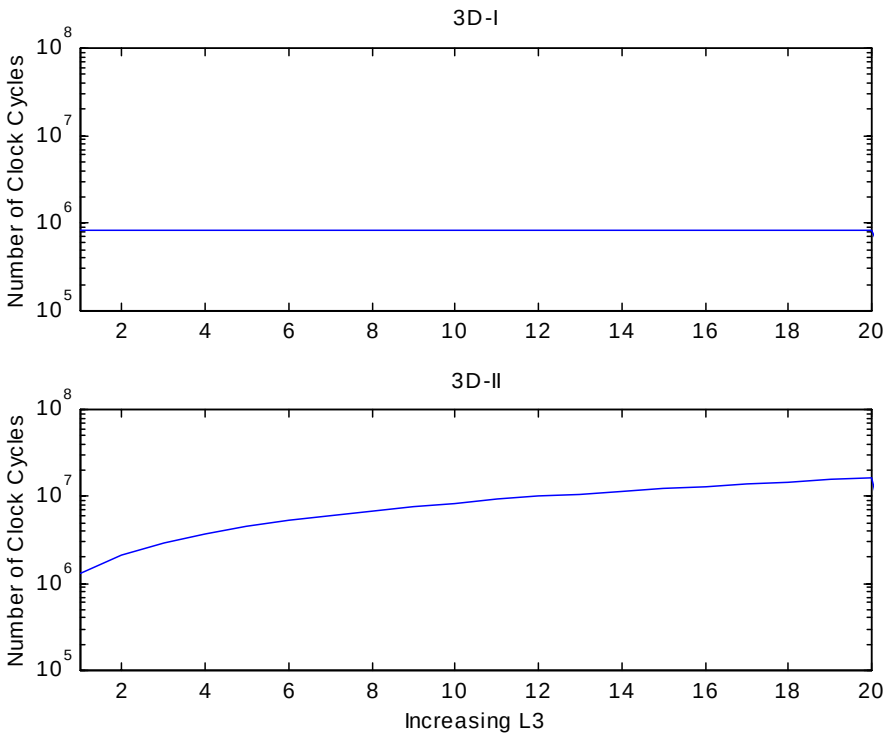


Figure 5.5 - Time Requirement (in Clock Cycles) for Filter Size L_3

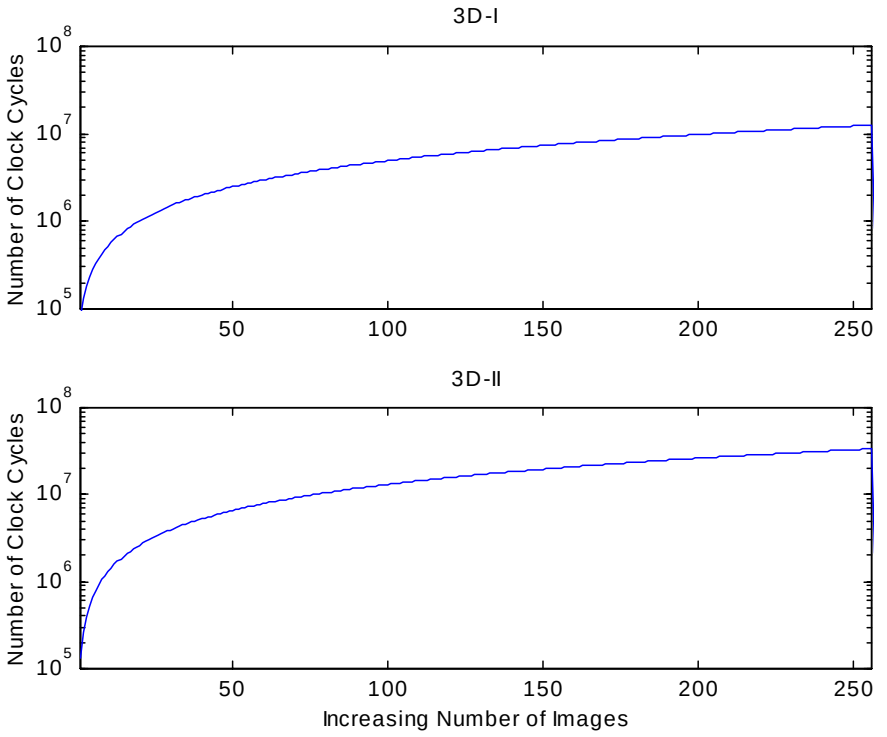


Figure 5.6 - Time Requirement (in Clock Cycles) for Number of Images

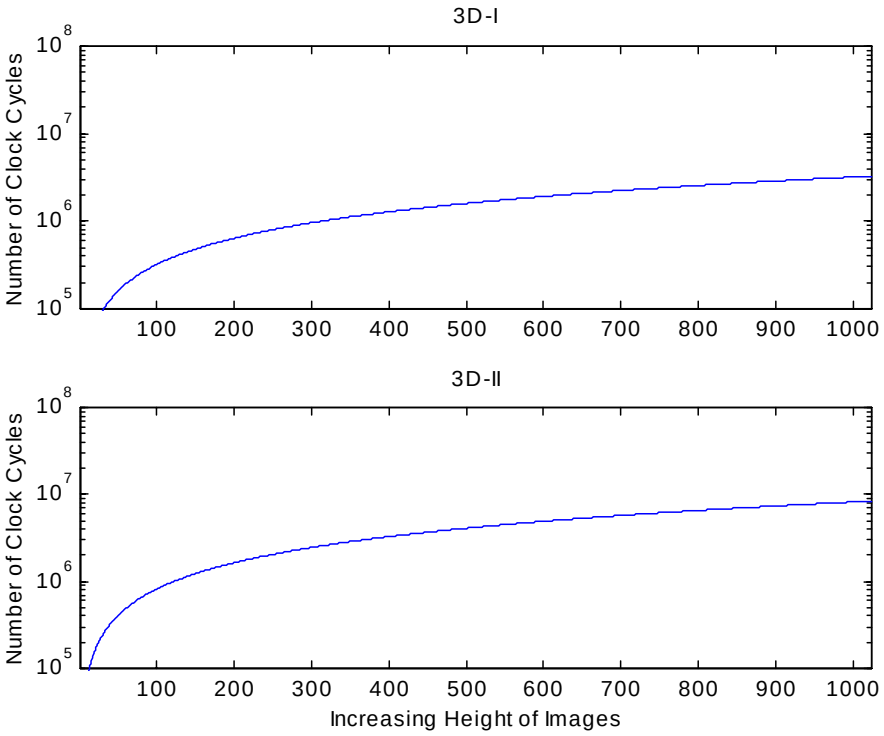


Figure 5.7 - Time Requirement (in Clock Cycles) for Height of Images

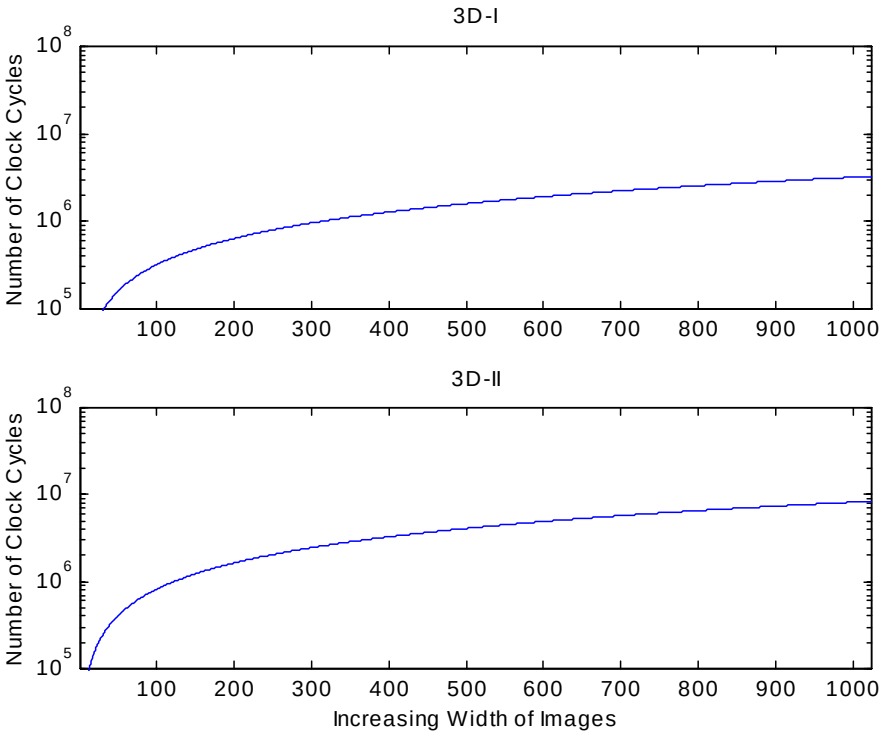


Figure 5.8 - Time Requirement (in Clock Cycles) for Width of Images

5.11 SPEED

Current 1-D DWT processing chips have speeds ranging from 5 MHz (Gonzalez) to 30 MHz (Aware), table 5.3. Intel has Pentium chips available that perform at speeds of 200 MHz and higher [Intel97]. At this speed, both architectures would be able to perform 1 octave of the 3-D DWT on a sequence of images without a problem. However, the architectures are able to perform at even much slower clock frequencies, as the graphs below show. The speed of each design is compared to an increasing number of images. In other words, at the standard rates of 30 frames per second and 60 frames per second, how fast do the 3D DWT transformers need to run? This question is not simple to answer, since the frames per second depend on both image size and filter lengths.

Design	Clock Frequency
Gonzalez 1-D [Gonzalez96]	5 MHz
Knowles 1-D [Knowles90]	6 MHz
Grzeszczak 1-D [Grzeszczak96]	20 MHz
Sheu 1-D [Sheu96]	20 MHz
Yu 1-D [Yu98]	25 MHz
Aware 1-D [Aware94]	30 MHz

Table 5.3 - Speed of Current DWT Chips

Below in figures 5.9 through 5.16, the speed of the architectures is shown based on assumed clock frequencies. The graphs show how fast the chips would need to run to guarantee 30 fps. Figures 5.9-5.12 show the performance based on a small wavelet transform, 4x4x2. A larger wavelet transform of 12x12x4 was chosen for figures 5.13-5.16. The image sizes progress in these graphs, from the small size of MRI data (256x256), to a larger 512x512, then 720x1280

and finally 1080x1290. The last two sizes were chosen to be compatible with HDTV. Tables 5.4 and 5.5 list the lowest operating frequency for the two architectures to achieve 30 fps and 60 fps. Remember that these results are based on one of each chip. Two 3D-II chips running in parallel only need to operate at half the clock rates listed below. These tables come from the frames per second equations:

$$\text{3D-I fps} = (4*\text{Speed} - 2NM - 2(L_1 + L_2 + L_3)) / (NM + 2NM)$$

$$\text{3D-II fps} = 8*\text{Speed} / (NM*(L_2L_3 + 2L_3 + 4))$$

Image size:	256x256		512x512		720x1280		1080x1920	
FPS:	30	60	30	60	30	60	30	60
3D-I MHz	2	3	6	12	21	42	48	94
3D-II MHz	4	8	16	31	55	111	124	249

Table 5.4 - Minimum Clock Speed (in MHz) for 3D-I and 3D-II with a 4x4x2 Wavelet

Image size:	256x256		512x512		720x1280		1080x1920	
FPS:	30	60	30	60	30	60	30	60
3D-I MHz	2	3	6	12	21	42	48	94
3D-II MHz	15	29	59	118	207	415	467	933

Table 5.5 - Minimum Clock Speed (in MHz) for 3D-I and 3D-II with a 12x12x4 Wavelet

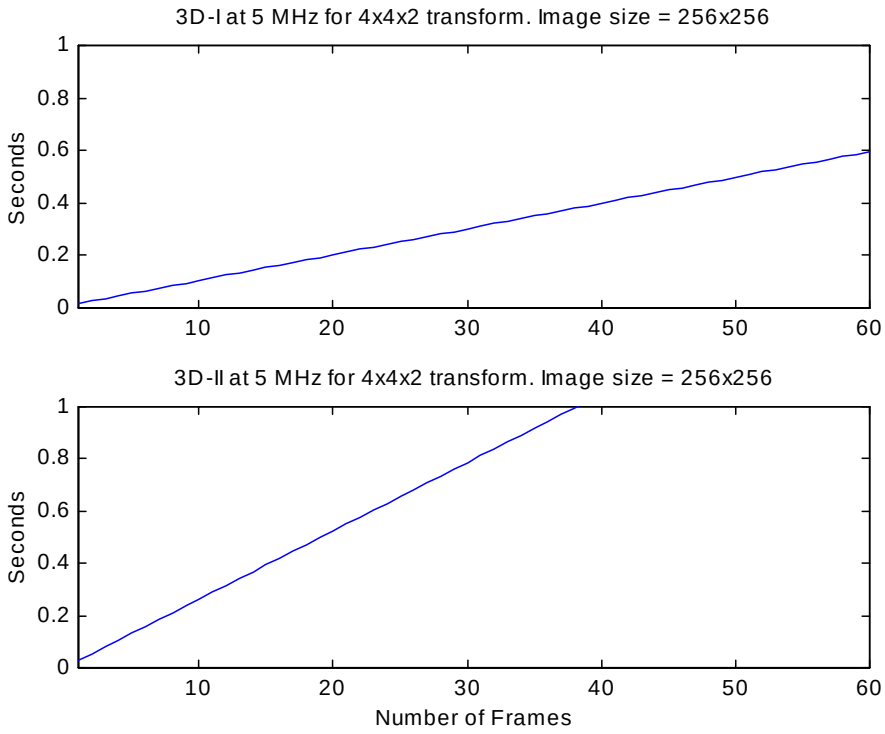


Figure 5.9 - Frames Per Second Analysis for 4x4x2 Transform and 256x256 Image

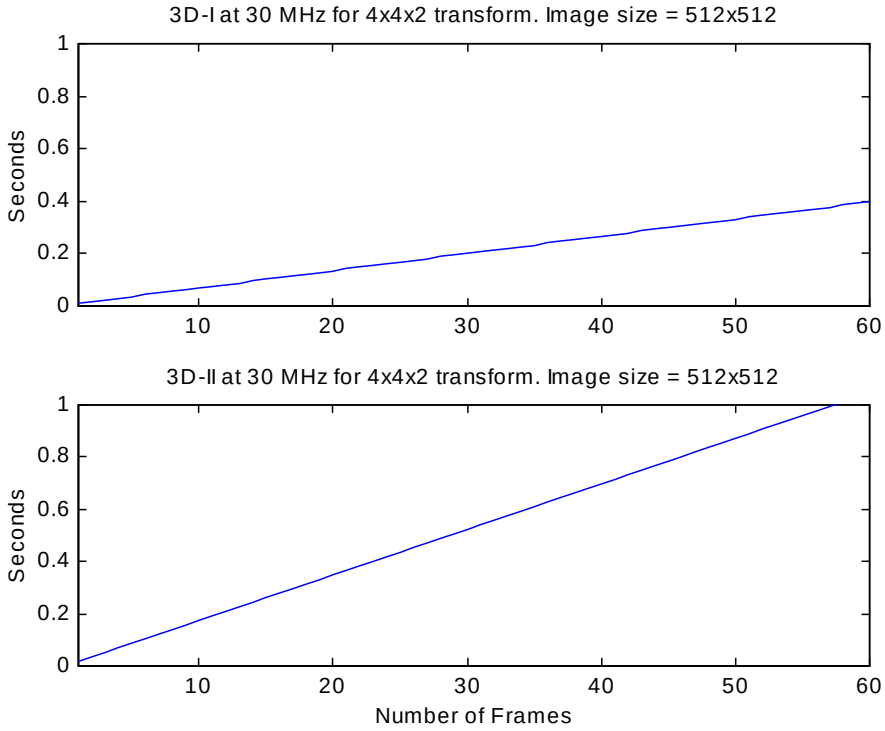


Figure 5.10 - Frames Per Second Analysis for 4x4x2 Transform and 512x512 Image

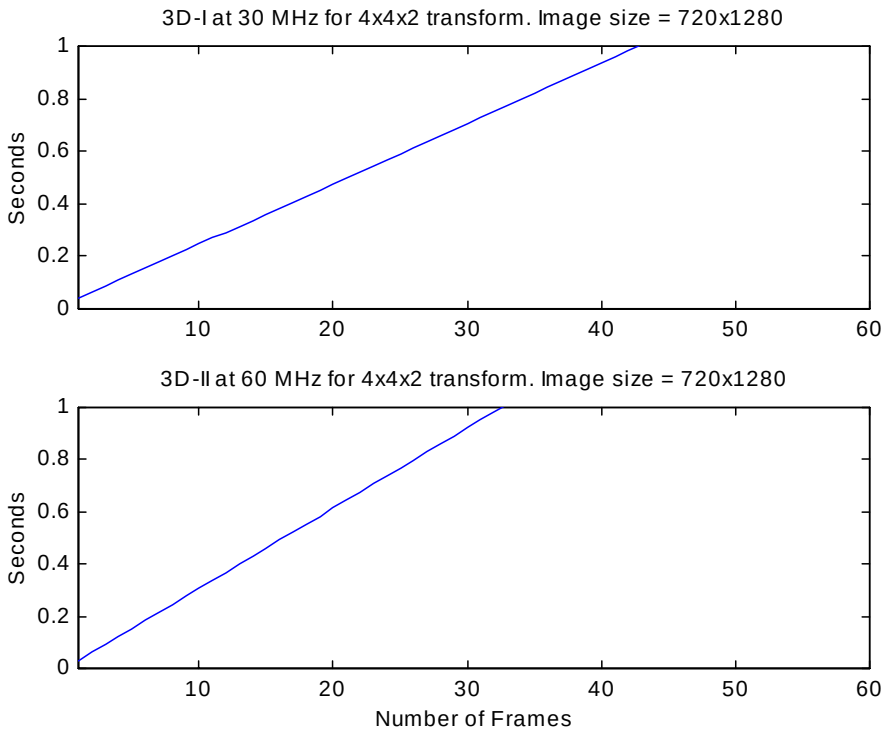


Figure 5.11 - Frames Per Second Analysis for 4x4x2 Transform and 720x1280 Image

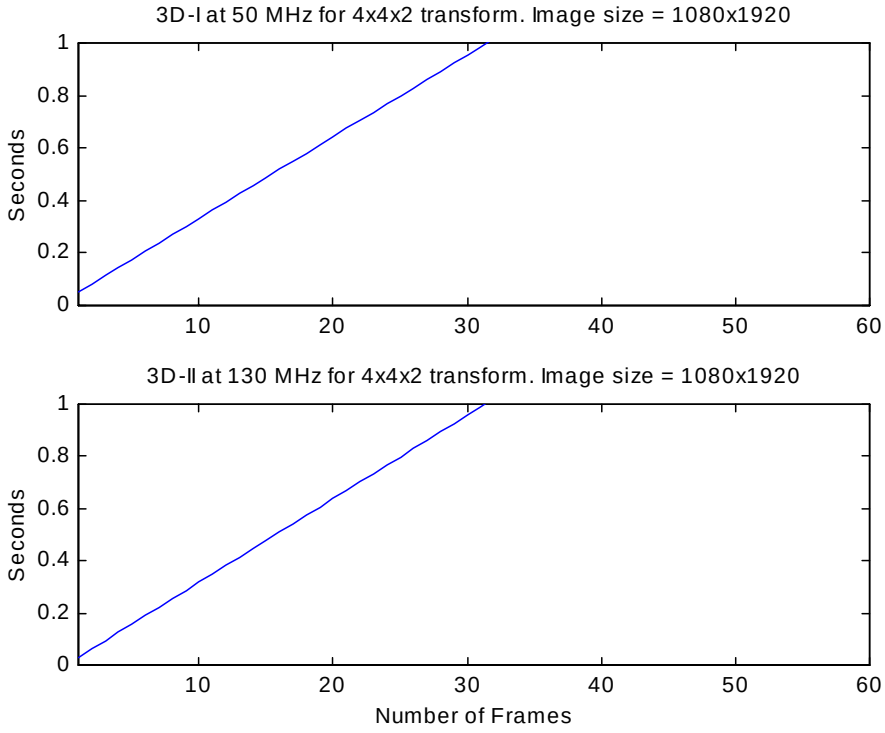


Figure 5.12 - Frames Per Second Analysis for 4x4x2 Transform and 1080x1920 Image

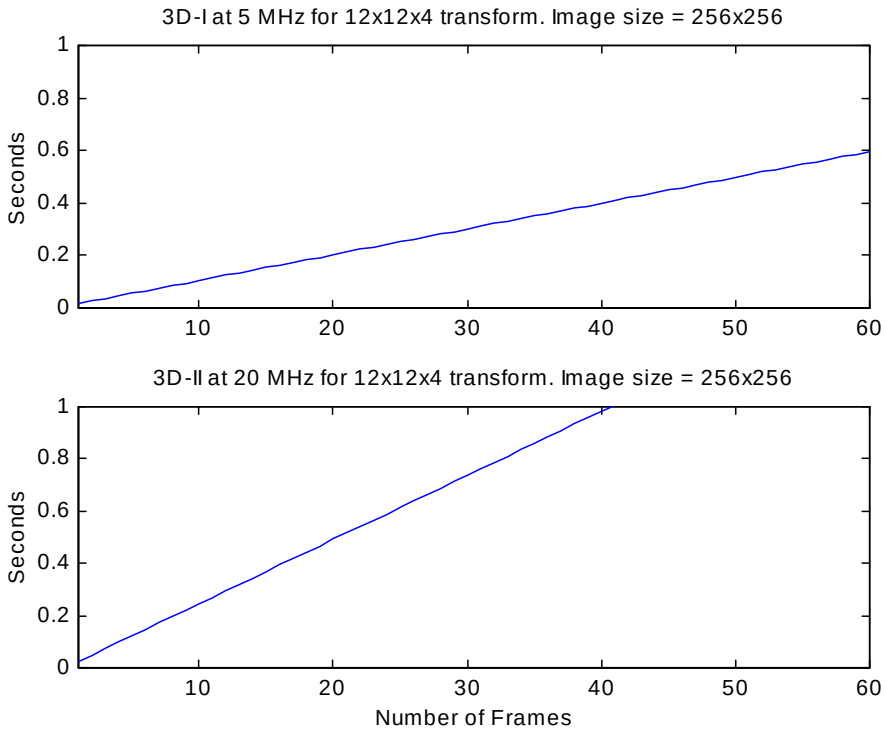


Figure 5.13 - Frames Per Second Analysis for 12x12x4 Transform and 256x256 Image Size

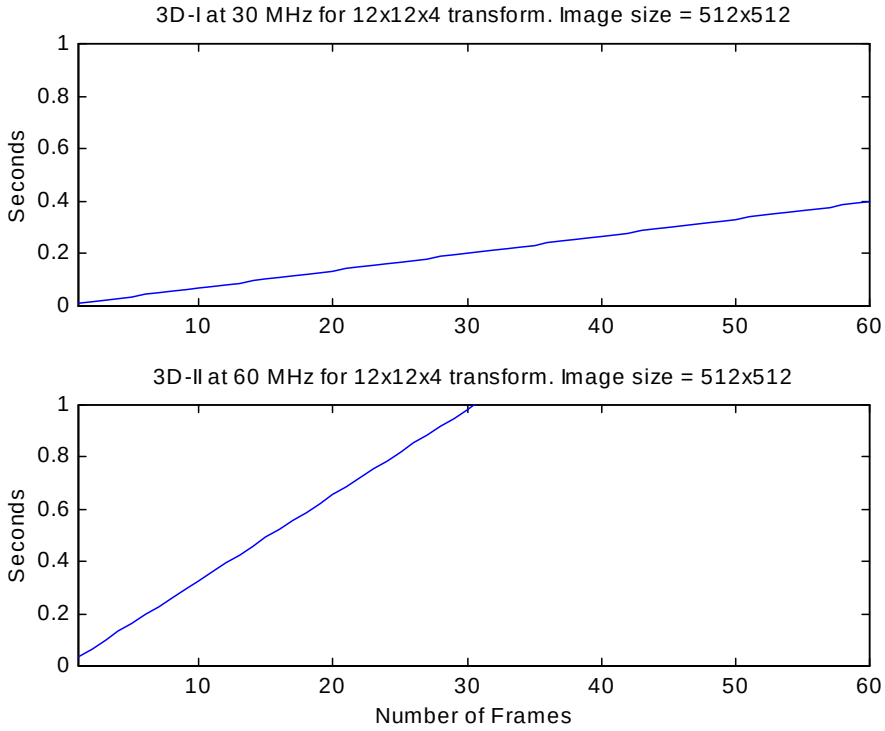


Figure 5.14 - Frames Per Second Analysis for 12x12x4 Transform and 512x512 Image Size

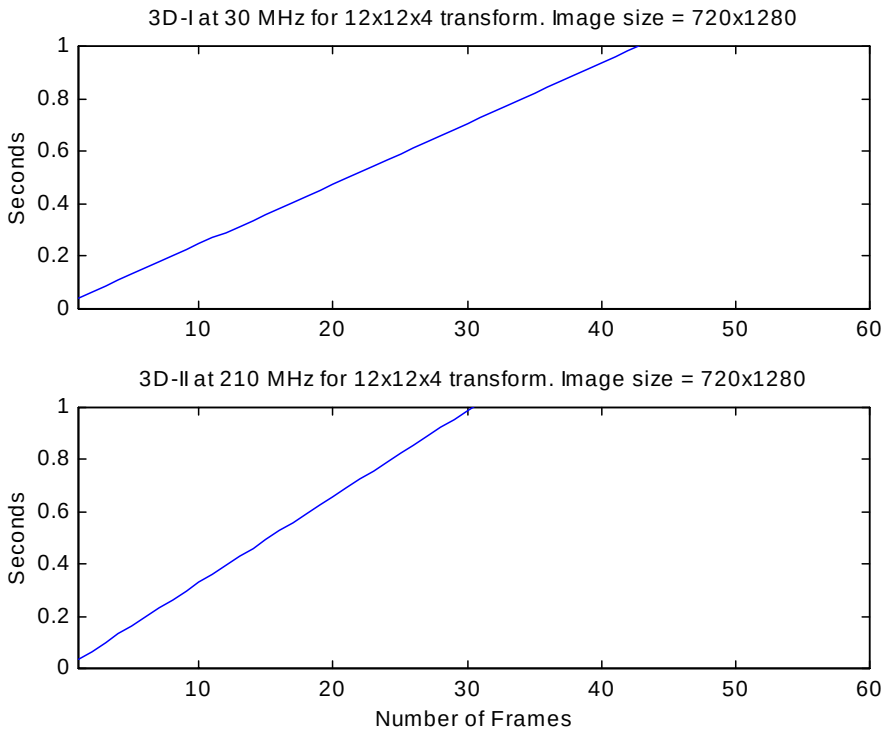


Figure 5.15 - Frames Per Second Analysis for 12x12x4 Transform and 720x1280 Image Size

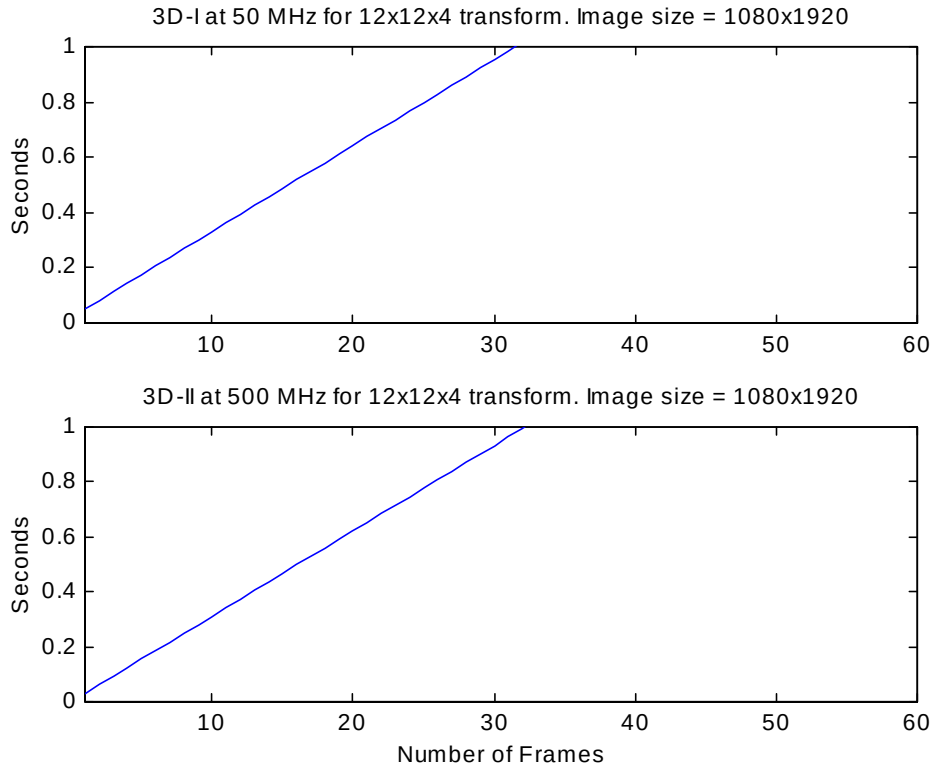


Figure 5.16 - Frames Per Second Analysis for 12x12x4 Transform and 1080x1920 Image Size

5.12 MEMORY REQUIREMENTS

Figure 5.17 shows the on-chip memory requirement for the two architectures, which can be expressed as follows.

$$\text{Memory (3D-I)} = 2*N*M*P + 2*N*M + 2*(L_1+L_2+L_3) \text{ words}$$

$$\text{Memory (3D-II)} = L_1*L_2*L_3 + 2*L_2*L_3 + 4*L_3 \text{ words}$$

The word width equals the amount of precision used (e.g. 16 bits). As figure 5.17 shows, changing the data size has a direct effect on the amount of memory needed by 3D-I, but no effect on the memory needed by 3D-II. The amount needed by 3D-II is dwarfed by the amount needed by 3D-I from the start. The memory requirements of the 2 architectures, when the wavelets increase in size, are shown in figure 5.18. The L_3 wavelet was chosen to vary since it has the most dramatic effect on the 3D-II memory requirement. The other L_1 and L_2 wavelets are chosen to be large if not impractical. Both of these wavelets are assumed to be 20 taps long, when [Fridman97] notes that wavelets typically are not longer than 12. However, even with large wavelets, the amount needed by 3D-I is still well above 3D-II, figure 5.18. In fact, simulating the memory needs with L_1 and L_2 wavelets of 100 taps each, the L_3 wavelet would have a width of 425 taps before the memory needed by 3D-II would surpass the memory needed by 3D-I. The other extreme, to equalize the memories, is to make the data size trivial, which is not realistic. Either way, the 3D-I architecture will always need several magnitudes more of on-chip memory than the 3D-II architecture for any realistic application.

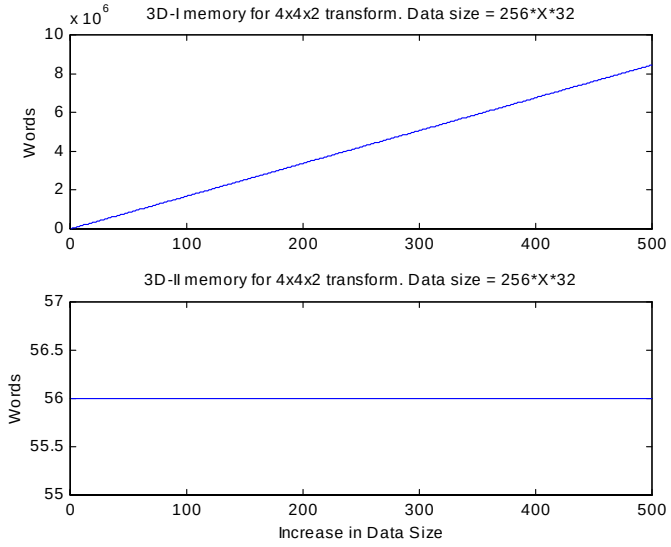


Figure 5.17 - On-Chip Memory Requirements versus Data Size

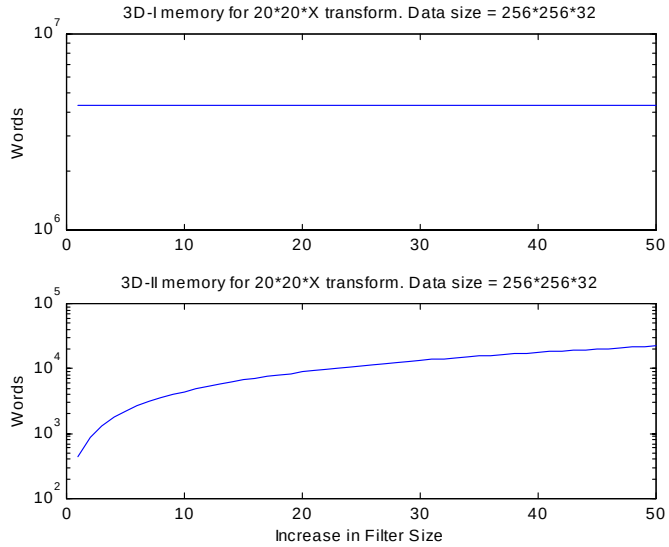


Figure 5.18 - On-Chip Memory Requirements versus Wavelet Filter Size

5.13 FOLDING ANALYSIS

Simulating the 3D-I architecture revealed that an extra latch is necessary for the odd inputs. MACs have 4 registers worth of storage in them in order to get the results and input to flow correctly. Otherwise, the results would enter the next MAC at the same time as input data, and the multiplier's output would be added to the wrong results. The MAC structure with the timing registers appears below in figure 5.19.

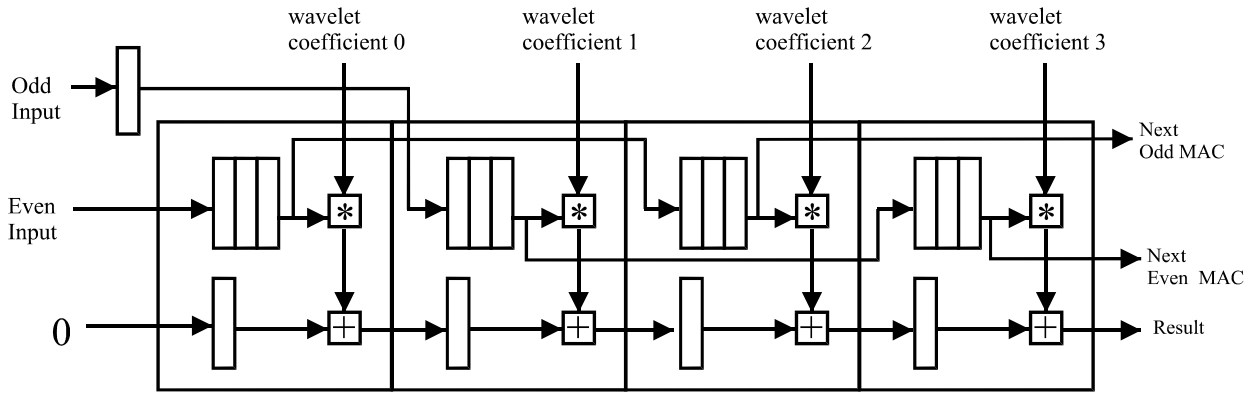


Figure 5.19 - The MAC Structure with Registers

This figure shows a 4 MAC sequence, but the register ratio is correct no matter how many taps are in the filter. The next problem to examine is how the MAC structure would change when folded. In other words, how many additional registers need to be added to allow the MACs to alternate between two data streams? Doubling the amount of registers in each MAC is the obvious solution. Experiments confirmed this assumption, figure 5.20. Note that extra padding is needed for the odd input as well.

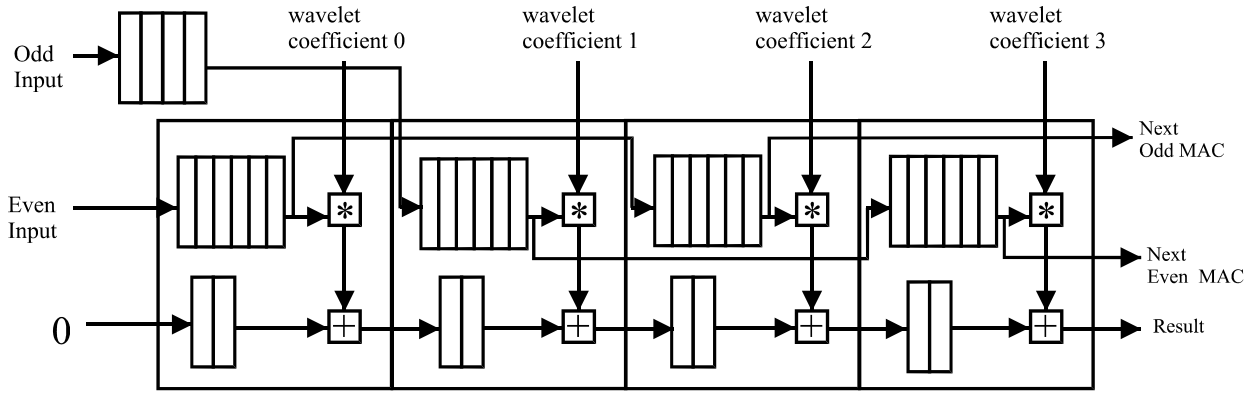
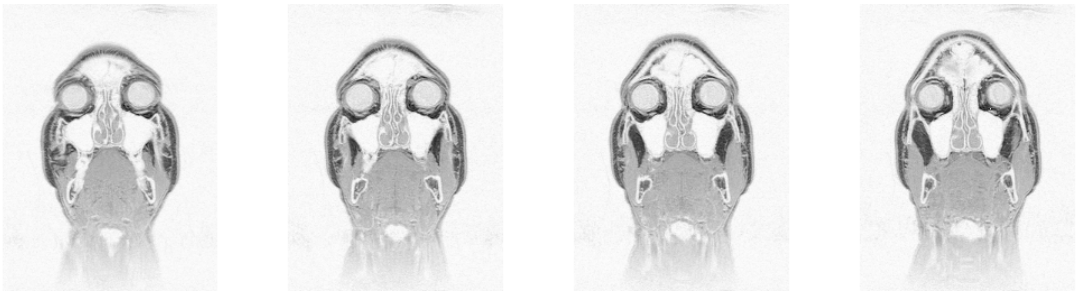


Figure 5.20 - The Revised MAC Structure for Folding

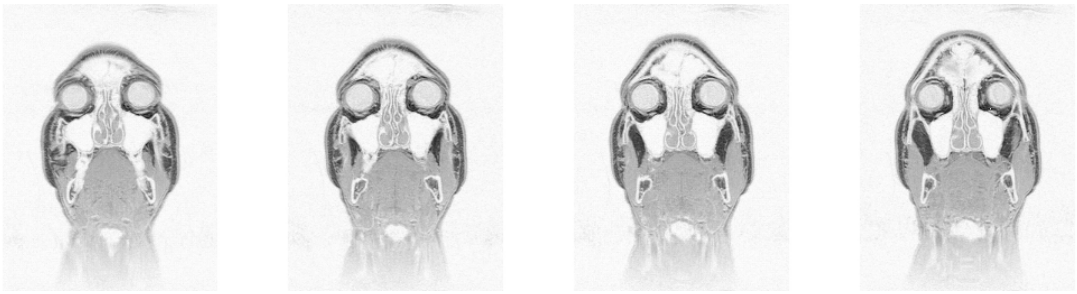
5.14 COMPRESSION RESULTS

The following experiments analyzed 3-D MRI data with different wavelets and compressed the results. Design decisions about the architecture, such as precision, affect the results. Therefore, the simulation takes the 3D-I and 3D-II architecture characteristics into account. The low-low-low pass results were stored in 16 bit integer format, while the detail signals were quantized and stored in 8 bit integer format. Low-low-low pass results have a range of -32768 to 32767, while the detail signals have a maximum range of -128 to 127. Figures 5.21, 5.22, and 5.23 show the uncompressed results for the first 4 slices, which are indistinguishable from the originals. The figures give the wavelet, the compression ratio, and the resulting PSNR. In the wavelet type, 2 represents a Haar (2 tap) wavelet, 4 stands for a 4 tap Daubechies wavelet, 8 means an 8 tap Daubechies wavelet, and 9,7 represents a biorthogonal spline. The figures also give the number of octaves. Note that the compression results from a 3-octave analysis has indistinguishable results, but the peak signal to noise ratios are higher than a comparable 1-octave compression.



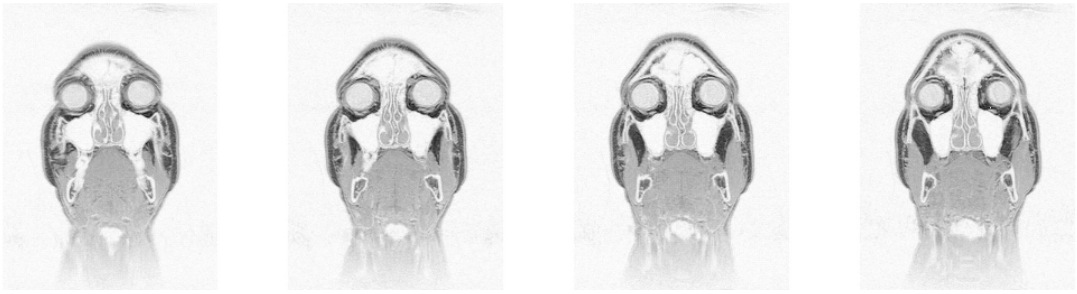
1 2 3 4

Original



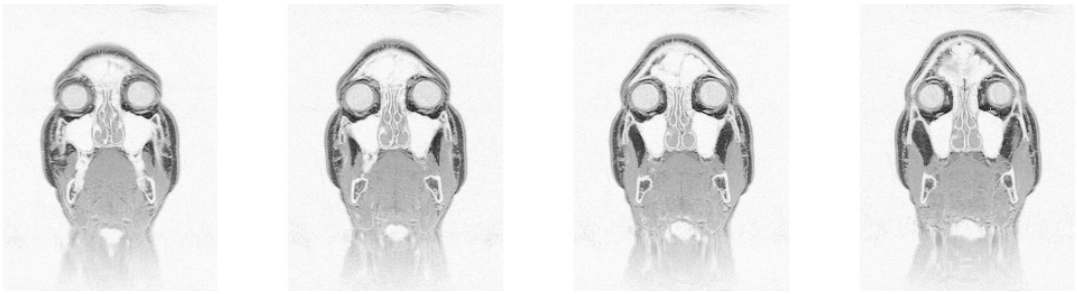
1 2 3 4

Wavelet: 4x4x2 1 Octave Compression ratio: 1.51 PSNR: 102.9



1 2 3 4

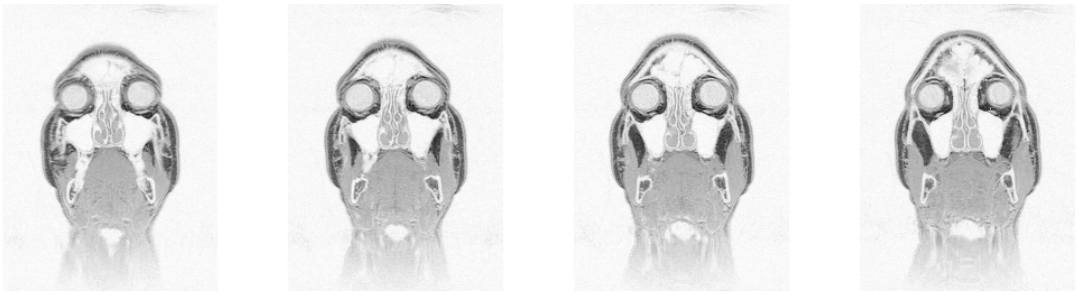
Wavelet: 8x8x2 1 Octave Compression ratio: 1.70 PSNR: 102.9



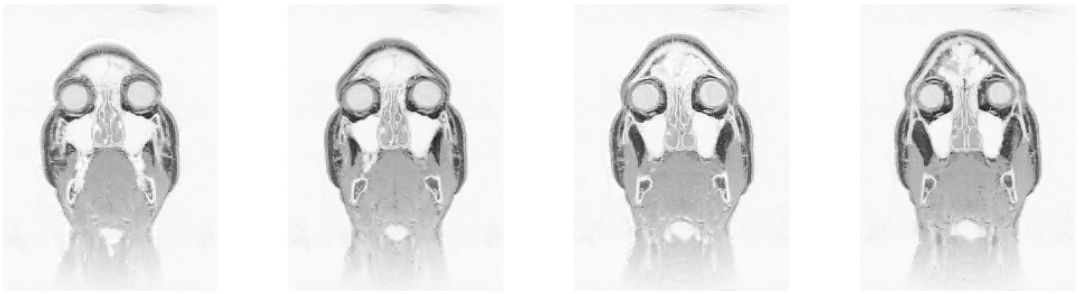
1 2 3 4

Wavelet: 9,7x9,7x2 1 Octave Compression ratio: 1.71 PSNR: 102.8

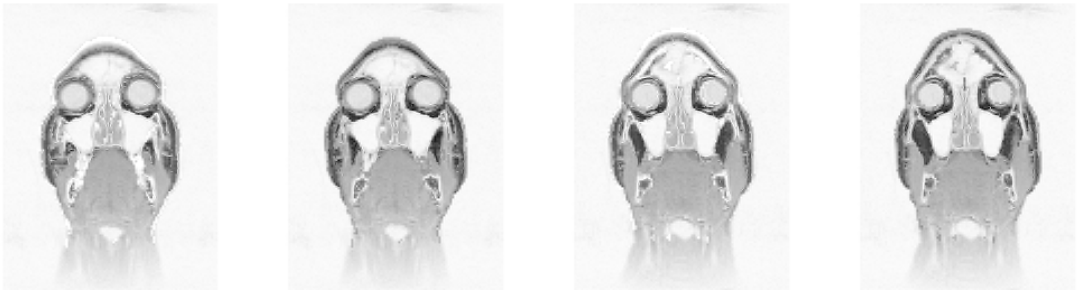
Figure 5.21 - Decompression Results 1



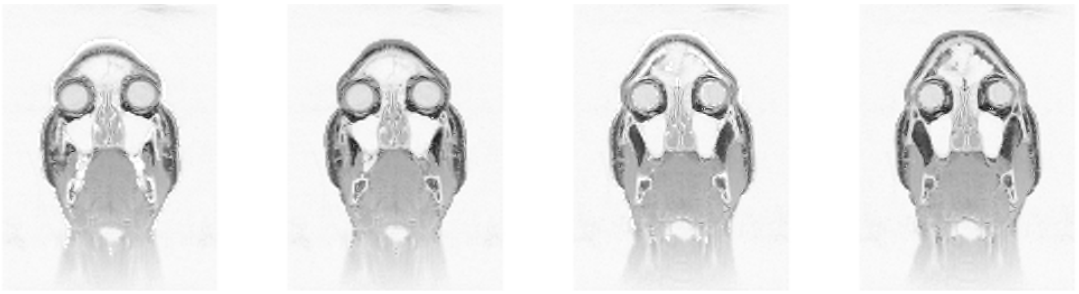
1 2 3 4
Wavelet: 4x4x4 1 Octave Compression ratio: 1.71 PSNR: 103.0



1 2 3 4
Wavelet: 4x4x2 1 Octave Compression ratio: 6.3 PSNR: 87.5

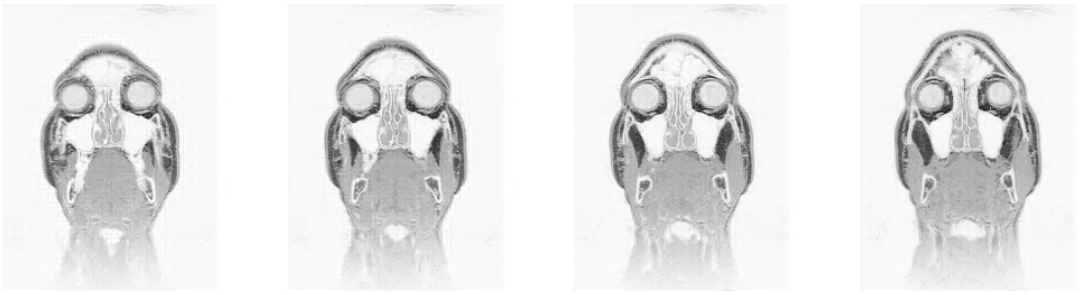


1 2 3 4
Wavelet: 8x8x2 1 Octave Compression ratio: 8.0 PSNR: 84.9

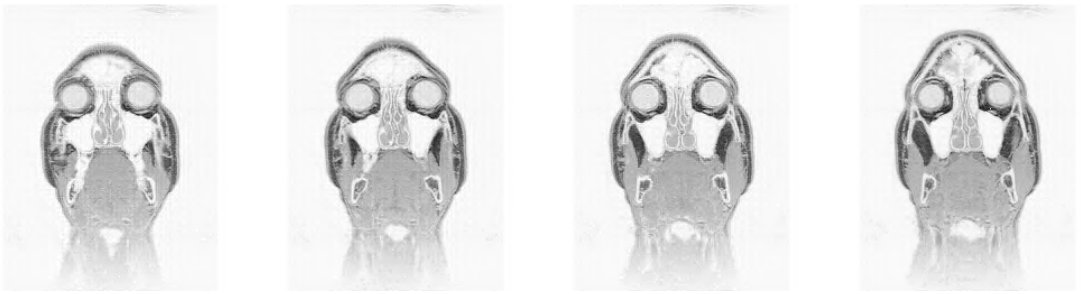


1 2 3 4
Wavelet: 9,7x9,7x2 1 Octave Compression ratio: 7.4 PSNR: 85.3

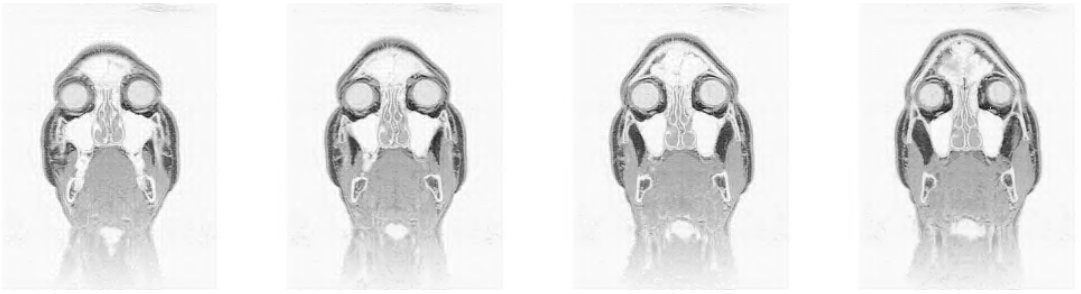
Figure 5.22 - Decompression Results 2



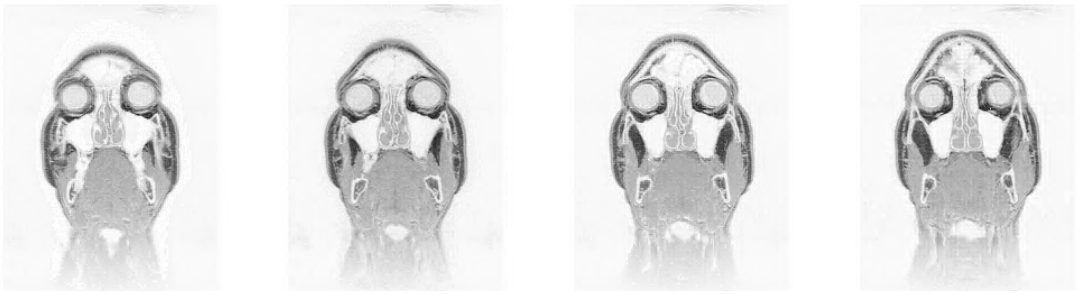
1 2 3 4
Wavelet: 4x4x2 3 Octaves Compression ratio: 8.5 PSNR: 213.0



1 2 3 4
Wavelet: 8x8x2 3 Octaves Compression ratio: 9.1 PSNR: 213.2



1 2 3 4
Wavelet: 9,7x9,7x2 3 Octaves Compression ratio: 9.8 PSNR: 213.6



1 2 3 4
Wavelet: 4x4x4 3 Octaves Compression ratio: 8.1 PSNR: 212.6

Figure 5.23 - Decompression Results 3

6. CONCLUSIONS

The Discrete Wavelet Transform can be used on 1, 2, and 3-dimensional 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. Presented in this work are two architectures to accomplish the 3-D DWT, the 3D-I and the 3D-II.

The 3D-I architecture is a straightforward implementation of the 3-D DWT. It allows even distribution of the processing load onto 3 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 Y and Z 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.

The problems of the first architecture are memory requirements and latency. It requires 2*N*M*P + 2*M*N words for intermediate data storage. While 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 Z filters cannot start until N*M*P/4 + N*M/2 clock cycles,

which means it takes N*M*P + N*M/2 + L_3/2 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 multi-dimensional serial architectures.

The 3D-II architecture has a single low/high filter pair to compute all the outputs. It requires a small amount of storage, based on the filter size, O(L_1xL_2xL_3), and not the input size. The filter sizes are much smaller than the input size (L_i << N). The latency is small, it depends on the filter sizes and not the input size. For example, using a 4x4x2 wavelet, the first output comes at the 12th clock cycle. The architecture can be used in parallel to transform the data in half the time (or less, if more than 2 are used). Complex control and the large number of clock cycles are the major drawbacks. Also, a large off-chip buffer is needed to allow block inputs.

The 3D-II architecture's need for a fast clock rate can be offset by having multiple 3D-II chips running in parallel. If the original data were divided into two sets, each set containing roughly half the frames, two chips could process the data in half the time needed. The first set would need to have L_3-1 extra frames, while the second set would not need any extra (unless circular inputs are desired). Similarly, the data could be split into many sets and sent to multiple 3D-II chips. Though the 3D-II uses many clock cycles for a large wavelet on a large data volume, putting multiple chips in parallel will allow each chip to run at a fraction of the speed that would be required of one chip.

Table 6.1 summarizes the two architectures, and shows how they compare to each other.

3D-I 3D-II

Memory	very large based on input	very small based on filters
Cascadable	yes	no
Runs in Parallel	not easily	easily
Design	Folded	Overlapping Block
Scalable to a larger wavelet	yes (easily)	yes (moderately difficult)
Size	large (due to memory)	small
Type of Filters	Serial (semi-systolic)	Parallel
Control	Simple, distributed	Complex, centralized
Number of Clock Cycles	low	high
Number of MACS	$2*(L_1+L_2+L_3)$	-
Number of Multipliers	-	$2*\max(L_1+L_2+L_3)$
Number of Adders	-	$5 + 2*(\max(L_1+L_2+L_3)-1)$
Latency	long	short

Table 6.1 - Comparison of the 3D-I and 3D-II architectures

In conclusion, this dissertation presents 2 architectures for the 3-D DWT. The DWT works well for compression, among other uses. Three dimensional data, such as medical applications, need a true 3-D transform. These are two of the first architectures to perform the 3-D Discrete Wavelet Transform. Each architecture has advantages and disadvantages over the other, such as memory requirements, latency, and frames per second. Therefore, which architecture to use depends upon the application parameters. For example, a television application has fixed dimensions on the data, and needs the results as fast as possible. Therefore,

the 3D-I architecture suits it well. A storage and transmission system for a hospital might want to use the same encoder for MRI, Positron Emission Tomography, and Computerized Tomography data, each with different data dimensions. The 3D-II architecture allows one device to handle all three different data sets. Thus, both architectures have potential uses.

REFERENCES

[Acharya97] Tinku Acharya, Po-Yueh Chen, and Hamid Jafarkhani, "A Pipelined Architecture for Adaptive Image Compression using DWT and Spectral Classification," *Proceedings of the Thirty-First Annual Conference on Information Sciences and Systems*, Gerard G. L. Meyer, Howard L. Weinert (editors), Volume II, The Johns Hopkins University, Baltimore, Maryland, March 19-21, 1997, pages 1-17.

[Aditya96] Sri-Krishna Aditya, *Fast Algorithm and Architecture for Computation of Discrete Wavelet Transform*, Ph. D. Dissertation, University of Southwestern Louisiana, 1996.

[Angelidis94] P. A. Angelidis, “MR Image Compression using a Wavelet Transform Coding Algorithm," *Magnetic Resonance Imaging*, Volume 12, Number 7, 1994, pages 1111-1120.

[Anton91] Howard Anton, *Elementary Linear Algebra*, 6th Edition, John Wiley & Sons, Inc., New York, 1991.

[Aware94] *Wavelet Transform Processor Chip User's Guide*, Aware, Inc., Bedford, MA, 1994.

[Benton95] Ryan Benton, Afshin Ganjoo, Beth Lumetta, Daryl Spillman, and Jason Ring, "Adaptive Wavelet Transforms of Singular and Chaotic Signals," *SPIE Wavelet Applications III* [Society of Photo-Optical Instrumentation Engineers], Harold H. Szu (editor), Volume 2762, Orlando, Florida, April 8-12 1996, pages 136-143.

[Bogner75] R. E. Bogner and A. G. Constantinides (editors), *Introduction to Digital Filtering*, John Wiley & Sons, Ltd., London, 1975.

[Boles93] W. W. Boles and Quang Minh Tieng, "Recognition of 2-D Objects from the Wavelet Transform Zero-crossing Representation," *Proceedings SPIE* [Society of Photo-Optical Instrumentation Engineers], A. F. Laine (editor), Volume 2034, Mathematical Imaging, San Diego, July 11-16, 1993, pages 104-114.

[Brislawn96] Christopher M. Brislawn, Jonathan N. Bradley, Remigius J. Onyshczak, and Tom Hopper, “The FBI Compression Standard for Digitized Fingerprint Images," *Proceedings SPIE* [Society of Photo-Optical Instrumentation Engineers], Andrew G. Tescher (editor), Volume 2847, Denver, August 4-9, 1996, pages 344-355.

[Brooks96] Geoffrey Brooks, "Processors for Wavelet Analysis and Synthesis: NIFS and the TI-C80 MVP," *Proceedings SPIE* [Society of Photo-Optical Instrumentation Engineers], H. H. Szu (editor), Volume 2762, Orlando, Florida, April 8-12, 1996, pages 325-333.

[Bruce96] Andrew Bruce, David Donoho, and Hong-Ye Gao, "Wavelet Analysis," *IEEE Spectrum*, October 1996, pages 26-35.

[Chakrabarti95] Chaitali Chakrabarti and Mohan Vishwanath, "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers," *IEEE Transactions on Signal Processing*, Volume 43, Number 3, March 1995, pages 759-771.

[Chakrabarti96] Chaitali Chakrabarti, Mohan Vishwanath, and Robert Owens, "Architectures for Wavelet Transforms: A Survey," *Journal of VLSI Signal Processing*, Volume 14, Number 1, 1996, pages 171-192.

[Chang97] Charles C. Chang, Jyh-Charn Liu, and Andrew K. Chan, "On the Architectural Support for Fast Wavelet Transform," *SPIE Wavelet Applications IV* [Society of Photo-Optical Instrumentation Engineers], H. H. Szu (editor), Volume 3078, Orlando, Florida, April 21-25, 1997, pages 700-707.

[Chen95] Jijun Chen and Magdy Bayoumi, "A Scalable Systolic Array Architecture for 2-D Discrete Wavelet Transforms," *Proceedings of IEEE Workshop on VLSI Signal Processing*, Jan Rabaey (editor), Volume III, Osaka, Japan, October 16-18, 1995, pages 303-312.

[Chinnock95] Chris Chinnock, "Wavelets Challenge MPEG, Fractals," *Byte*, Volume 20, Issue 12, December 1995, page 34.

[Chou93] Mo-Hong Chou and En-Bing Lin, "Wavelet Compression Through Wavelet Stieltjes Transform," *Mathematical Imaging: Wavelet Applications in Signal and Image Processing, SPIE Proceedings* [Society of Photo-Optical Instrumentation Engineers], Andrew F. Laine (chair/editor), Volume 2034, San Diego, July 15-16, 1993, pages 254-264.

[Coifman93] R. R. Coifman and M. V. Wickerhauser, "Wavelets and Adapted Waveform Analysis," *Proceedings of Symposia in Applied Mathematics*, Ingrid Daubechies (Editor), Volume 47, American Mathematics Society, Providence, November 6-9 1993, pages 119-153.

[Daubechies92] Ingrid Daubechies, *Ten Lectures on Wavelets*, Capital City Press, Montpelier, Vermont, 1992.

[Edwards92] Tim Edwards, "Discrete Wavelet Transforms: Theory and Implementation," unpublished paper presented at Stanford University, June 4, 1992.

[Fridman94a] Jose Fridman and Elias S. Manolakos, "Distributed Memory and Control VLSI Architectures for the 1-D Discrete Wavelet Transform," *IEEE Proceedings VLSI Signal Processing VII*, Jan Rabaey, Paul M. Chau, John Eldon (editors), La Jolla, California, October 26-28, 1994, pages 388-397.

[Fridman94b] Jose Fridman and Elias S. Manolakos, "On the Synthesis of Regular VLSI Architectures for the 1-D Discrete Wavelet Transform," *Proceedings of SPIE Conf. on Mathematical Imaging: Wavelet Applications in Signal and Image Processing II* [Society of Photo-Optical Instrumentation Engineers], Andrew F. Laine, Michael A. Unser (editors), Volume 2303, San Diego, July 24-29, 1994, pages 91-104.

[Fridman97] Jose Fridman and Elias S. Manolakos, "Discrete Wavelet Transform: Data Dependence Analysis and Synthesis of Distributed Memory and Control Array Architectures," *IEEE Transactions on Signal Processing*, Volume 45, Number 5, May 1997, pages 1291-1308.

[Gibbs96] W. Wayt Gibbs, "Making Wavelets," *Scientific American*, June 1996, pages 137-138.

[Goh93] K. H. Goh, J. J. Soraghan, and T. S. Durrani, "New 3-D Wavelet Transform Coding Algorithm for Image Sequences," *Electronics Letters*, Volume 29, Number 4, 1993, pages 401-402.

[Gonzalez96] Gerardo Gonzalez-Altamirano, Alejandro Diaz-Sanchez, and Jaime Ramirez-Angulo, "Fast Sampled-Data Wavelet Transform CMOS VLSI Implementation," *Proceedings of the 39th Midwest Symposium on Circuits and Systems*, G. Cameron, M. Hassoun, A. Jerdee, C. Melvin (editors), Iowa State University, Ames, Iowa, August 18-21, 1996, pages 101-104.

[Graps95] Amara Graps, "An Introduction to Wavelets," *IEEE Computational Science and Engineering*, Volume 2, Number 2, 1995, pages 1-18.

[Grzeszczak96] Aleksander Grzeszczak, Mrinal K. Mandal, and Sethuraman Panchanathan, Tet Yeap, "VLSI Implementation of Discrete Wavelet Transform," *IEEE Transactions of Very Large Scale Integration (VLSI) Systems*, Volume 4, Number 4, 1996, pages 421-433.

[Henricks96] Mark Henricks, "PC Pictures Get the Wavelet Squeeze," *Popular Science*, Volume 248, Issue 2, February 1996, page 30.

[Hickman97] Angela Hickman, John Morris, Carol Levin, Sebastian Rupley, and David Willmott, "Web Acceleration," *PC Magazine*, June 10, 1997, page 10.

[Hilton94] Michael Hilton, Bjorn Jawerth, and Ayan Sengupta, "Compressing Still and Moving Images with Wavelets," *Multimedia Systems*, Volume 2, Number 3, 1994, pages 1-19.

[Intel97] *Pentium® Processor Family Developer's Manual*, Intel Corporation, Hillsboro, Oregon, 1997.

[Jennison61] R. C. Jennison, *Fourier Transforms and Convolutions for the Experimentalist*, Pergamon Press Ltd., Leipzig, German Democratic Republic, 1961.

[Kaiser94] Gerald Kaiser, *A Friendly Guide to Wavelets*, Birkhauser, Boston, 1994.

[Kim97] Jaemin Kim and John W. Woods, "Spatio-Temporal Adaptive 3-D Kalman Filter for Video," *IEEE Transactions on Image Processing*, Volume 6, Number 3, 1997, pages 414-424.

[Knowles90] G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform," *Electronics Letters*, Volume 26, Number 15, 1990, pages 1184-1185.

[Kotsas94] P. Kotsas, D. W. Piraino, M. P. Recht, and B. J. Richmond, "Comparison of Adaptive Wavelet-based and Discrete Cosine Transform Algorithm in Image Compression", *Radiology*, Volume 193(P), Supplement, 1994, page 331.

[Kwai96] Ding-Ming Kwai and Behrooz Parhami, "FFT Computation with Linear Processing Arrays Using a Data-Driven Control Scheme," *Journal of VLSI Signal Processing Systems*, Volume 13, Number 1, 1996, pages 57-66.

[Lankswert92] Patrick C. Lankswert, *Image Processing Tools and Methods Guide*, Department of Engineering Mathematics and Computer Science, University of Louisville, Louisville, Kentucky, August 1992.

[Lee93] Heesub Lee, Yongmin Kim, Alan H. Rowberg, and Eve A. Riskin, "Statistical Distributions of DCT Coefficients and Their Application to an Interframe Compression Algorithm for 3-D Medical Images," *IEEE Transactions of Medical Imaging*, Volume 12, Number 3, 1993, pages 478-485.

[Lewis90] A. S. Lewis and G. Knowles, "Video Compression using 3-D Wavelet Transforms," *Electronics Letters*, Volume 26, Number 6, 1990, pages 396-398.

[Lewis91] A. S. Lewis and G. Knowles, "VLSI Architecture for 2-D Daubechies Wavelet Transform without Multipliers," *Electronics Letters*, Volume 27, Number 2, 1991, pages 171-173.

[Lewis92] A. S. Lewis, G. Knowles, "Image Compression Using the 2-D Wavelet Transform," *IEEE Transactions on Image Processing*, Volume 1, Number 2, 1992, pages 244-250.

[Limqueco96a] Jimmy Limqueco and Magdy Bayoumi, "A 2-D DWT Architecture," *Proceedings of the 39th Midwest Symposium on Circuits and Systems*, G. Cameron, M. Hassoun, A. Jerdee, C. Melvin (editors), Iowa State University, Ames, Iowa, August 18-21, 1996, pages 1239-1242.

[Limqueco96b] Jimmy Limqueco and Magdy Bayoumi, "A Scalable Architecture for 2-D Discrete Wavelet Transform," *VLSI Signal Processing IX*, Teresa Meng, Wayne Burleson (editors), San Francisco, October 30-November 1, 1996, pages 369-377.

[Loy88] Nicholas John Loy, *An Engineer's Guide to FIR Digital Filters*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[Mallat89] Stephane Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Pattern Analysis and Machine Intelligence*, Volume 11, Number 7, 1989, pages 674-693.

[Mallat91] Stephane Mallat, "Zero-Crossings of a Wavelet Transform," *IEEE Transactions on Information Theory*, Volume 17, Number 4, 1991, pages 1019-1033.

[Mallat92] Stephane Mallat and Sifen Zhong, "Characterisation of signals from multiscale edges," *IEEE Transactions PAMI* [Pattern Analysis and Machine Intelligence], Volume 14, Number 17, 1992, pages 710-732.

[Marr82] D. Marr, *Vision*, Freeman, New York, 1982.

[Matlab96] *Matlab Wavelet Handbook*, The MathWorks, Inc., Natick, Massachusetts, 1996.

[Mukherjee86] Amar Mukherjee, *Introduction to nMOS 7 CMOS VLSI System Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Oppenheim89] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[Ozer95] Jan Ozer, "New Compression Codec Promises Rates Close to MPEG," *CD-ROM Professional*, September 1995, page 24.

[Ozer96] Jan Ozer, "Web-Based Video - VDOLive" (product review), *PC Magazine*, Volume 15, Number 6, March 26, 1996, pages 136-137.

[Parhi93] Keshab K. Parhi and Takao Nishitani, "VLSI Architectures for Discrete Wavelet Transforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 1, Number 2, 1993, pages 191-202.

[Pratt96] Michael A. Pratt, C. Henry Chu, and Stephen Wong, "Volume Compression of MRI Data using Zerotrees of Wavelet Coefficients," *Proceedings Wavelet Applications in Signal and Image Processing IV*, Michael A. Unser, Akram Aldroubi, Andrew Laine (editors), Denver, Colorado, August 16-19, 1996, pages 752-763.

[Quarmby85] David Quarmby (editor), *Signal Processor Chips*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[Rayner71] J. N. Rayner, *An Introduction to Spectral Analysis*, Methuen, Inc., New York, 1971.

[Riskin90] Eve A. Riskin, Tom Lookabaugh, Philip A. Chou, and Robert M. Gray, "Variable Rate Vector Quantization for Medical Image Compression," *IEEE Transactions of Medical Imaging*, Volume 9, Number 3, 1990, pages 290-298.

[Serrano96] E. P. Serrano and M. A. Fabio, "Application of the Wavelet Transform to Acoustic Emission Signals Processing," *IEEE Transactions on Signal Processing*, Volume 44, Number 5, 1996, pages 1270-1275.

[Sheu96] Ming-Hwa Sheu, Ming-Der Shieh, and Shun-Fa Cheng, "A Unified VLSI Architecture for Decomposition and Synthesis of Discrete Wavelet Transform," *Proceedings of the 39th Midwest Symposium on Circuits and Systems*, G. Cameron, M. Hassoun, A. Jerdee, C. Melvin (editors), Iowa State University, Ames, Iowa, August 18-21, 1996, pages 113-116.

[Smith94] Emily T. Smith (editor), "Squeezing a Flood of Data into Waves," *Business Week*, Issue 3397, November 7, 1994, page 105.

[Strang95] Gilbert Strang and Bjorn Jawerth, *SIAM Short Course on Wavelets from Filter Banks*, SIAM [Society for Industrial and Applied Mathematics], Philadelphia, 1995.

[Syed95] Shafiullah Syed, Magdy Bayoumi, and Jimmy Limqueco, "An Integrated Discrete Wavelet Transform Array Architecture," *Proceedings of the Workshop on Computer Architecture for Machine Perception*, Virginio Cantoni (editor), Como, Italy, September 18-20, 1995, pages 32-36.

[Szu94] Harold Szu, Charles Hsu, Pradip Thaker, and Mona Zaghloul, "Image Wavelet Transforms Implemented by Discrete Wavelet Chips," *Optical Engineering*, Volume 33, Number 7, 1994, pages 2310-2325.

[Szu95] Harold Szu and Brian Telfer, "Wavelet Dynamics," *The Handbook of Brain Theory and Neural Networks*, Michael A. Arbib (editor), MIT Press, 1995, pages 1049-1053.

[Szu96] Harold H. Szu, "Review of Wavelet Transforms for Pattern Recognitions," *Wavelet Applications III, SPIE Proceedings* [Society of Photo-Optical Instrumentation Engineers], Harold H. Szu (chair/editor), Volume 2762, Orlando, Florida, April 8-12, 1996, pages 2-22.

[Szu97] Harold Szu and Charles Hsu, "Symmetric Codec Video Compression for N-to-N Teleconferences Using Biorthogonal Subband Wavelets," *SPIE Wavelet Applications IV* [Society of Photo-Optical Instrumentation Engineers], Harold H. Szu (chair/editor), Volume 3078, Orlando, Florida, April 22-24, 1997, pages 12-22.

[Taubman94] D. Taubman and A. Zakhor, "Multirate 3-D Subband Coding of Video," *IEEE Transactions on Signal Processing*, Volume 3, Number 1, 1994, pages 572-588.

[Tian96] Jun Tian, *Multiple Motion Compensated and Wavelet Based Video Compression*, Ph. D. Dissertation, University of Southwestern Louisiana, 1996.

[Villasenor93] John D. Villasenor, "Alternatives to the Discrete Cosine Transform for Irreversible Tomographic Image Compression," *IEEE Transactions of Medical Imaging*, Volume 12, Number 4, 1993, pages 803-811.

[Vetterli90] Martin Vetterli and Cormac Herley, "Wavelets and Filter Banks: Relationships and New Results," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Lonnie Ludeman (editor), Albuquerque, New Mexico, April 3-6, 1990, pages 1723-1726.

[Vetterli95] Martin Vetterli and Jelena Kovacevic, *Wavelets and Subband Coding*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1995.

[Vishwanath92] M. Vishwanath, R. M. Owens, and M. J. Irwin, "Discrete Wavelet Transforms in VLSI," *Proceedings of the International Conference on Application Specific Array Processors*, Jose Fortes, Edward A. Lee, Theresa H. Meng (editors), Berkeley, August 1-2, 1992, pages 218-229.

[Vishwanath94a] Mohan Vishwanath and Chaitali Chakrabarti, "A VLSI Architecture for Real-Time Hierarchical Encoding/Decoding of Video using the Wavelet Transform," *IEEE International Conference on Acoustics, Speech and Signal Processing*, Phil Plevin (editor), Adelaide, Australia, Volume 2, April 19-22, 1994, pages 401-404.

[Vishwanath94b] Mohan Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform," *IEEE Transactions of Signal Processing*, Volume 42, Number 3, 1994 pages 673-676.

[Vishwanath95] Mohan Vishwanath, Robert Owens, and Mary Irwin, "VLSI Architectures for the Discrete Wavelet Transform," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Volume 42, Number 5, 1995, pages 305-316.

[Walczak96] Beata Walczak, Bas van den Bogaert, and Desire' Luc Massart, "Application of Wavelet Packet Transform in Pattern Recognition of Near-IR Data," *Analytical Chemistry*, Volume 68, Number 1, 1996, pages 1742-1747.

[Wang94] Jun Wang and H. K. Huang, "Three-dimensional Wavelet Transform with Parallel Comp. for Radiological Image Compression", *Radiology*, Volume 193(P), Supplement, 1994, page 111.

[Wang95] Jun Wang and H. K. Huang, "Three-dimensional Medical Image Compression using a Wavelet Transform with Parallel Computing," *SPIE Imaging Physics* [Society of Photo-Optical Instrumentation Engineers], Yongmin Kim (editor), San Diego, Volume 2431, March 26-April 2, 1995, pages 16-26.

[Weste93] Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, Reading Massachusetts, 1993.

[Yazji97] Sausan Yazji, *A Motion-Estimation Architecture Based on Band- Matching*, M.S. Thesis, University of Southwestern Louisiana, 1997.

[Yu98] Chu Yu, Chien-An Hsieh, and Sao-Jie Chen, "Design and Realization of Two Efficient VLSI Architectures for Discrete Wavelet Transforms," submitted for publication.

APPENDIX A - SAMPLE WAVELET COEFFICIENTS

The wavelet coefficients are in floating point format, obtained from Matlab.

Haar
low pass filter coefficients = 0.7071 0.7071
high pass filter coefficients = -0.7071 0.7071
low pass reconstruction coefficients = 0.7071 0.7071
high pass reconstruction coefficients = 0.7071 -0.7071

Daubechies Wavelet Coefficients
db2
low pass = -0.1294 0.2241 0.8365 0.4830
high pass = -0.4830 0.8365 -0.2241 -0.1294
low reconstruction = 0.4830 0.8365 0.2241 -0.1294
hi reconstruction = -0.1294 -0.2241 0.8365 -0.4830

Coiflets
coif1
low pass = -0.0157 -0.0727 0.3849 0.8526 0.3379 -0.0727
high pass = 0.0727 0.3379 -0.8526 0.3849 0.0727 -0.0157
low reconstruction = -0.0727 0.3379 0.8526 0.3849 -0.0727 -0.0157
hi reconstruction = -0.0157 0.0727 0.3849 -0.8526 0.3379 0.0727

Symlets
sym2
low pass = -0.1294 0.2241 0.8365 0.4830
high pass = -0.4830 0.8365 -0.2241 -0.1294
low reconstruction = 0.4830 0.8365 0.2241 -0.1294
hi reconstruction = -0.1294 -0.2241 0.8365 -0.4830

Biorthogonal
bior4.4 (9,7 Biorthogonal Spline)
low pass = 0 0.0378 -0.0238 -0.1106 0.3774 0.8527 0.3774
-0.1106 -0.0238 0.0378
high pass = 0 -0.0645 0.0407 0.4181 -0.7885 0.4181 0.0407
-0.0645 0 0
low reconstruction = 0 -0.0645 -0.0407 0.4181 0.7885 0.4181 -0.0407
-0.0645 0 0
hi reconstruction = 0 -0.0378 -0.0238 0.1106 0.3774 -0.8527 0.3774
0.1106 -0.0238 -0.0378

The wavelet coefficients below have been converted from floating point to fixed point (16 bit) integer. The radix point in this example is after the 9th bit from the left.

For example:

Suppose we have the number 61.

61 = 0000 0000 0011 1101 (Convert to binary)

= 000000000.0111101 (Insert radix point)

= 1/4 + 1/8 + 1/16 + 1/32 + 1/128 (Convert to floating point)

= 0.4765625 (Floating point equivalent)

This means that the actual wavelet coefficient is 0.4765625, a floating-point number. But for our purposes, we just call it 124. If the number is greater than 32767, then it is negative (2's complement). To convert, simply subtract the number from 65536. For example, 65536 - 65503 = 33, which is the first Daubechies low pass coefficient (see below).
Notice how the same values repeat. This is really a subband filter with perfect reconstruction (refer to figure 1.1). We are implementing the analysis section for the 3-D DWT, which is a bit more complex, but the same idea holds.

Instead of multiplying by a floating point, which takes more time, the architectures multiply by integers with an assumed radix point.

Haar

low pass filter coefficients = 181 181 0 0
high pass filter coefficients = 65355 181 0 0
low pass reconstruction coefficients = 181 181 0 0
high pass reconstruction coefficients = 181 65355 0 0

Daubechies

db2

low = 65520 28 107 61
high = 65475 107 65508 65520
low reconstruction = 61 107 28 65520
high reconstruction = 65520 65508 107 65475

Coiflets

coif1

low = 65534 65527 49 109 43 65527
high = 9 43 65427 49 9 65534
low reconstruction = 65527 43 109 49 65527 65534
high reconstruction = 65534 9 49 65427 43 9

Symlets
sym2
low = 65520 28 107 61
high = 65475 107 65508 65520
low reconstruction = 61 107 28 65520
high reconstruction = 65520 65508 107 65475

Biorthogonal Spline
bior4.4
low = 0 4 65533 65522 48 109 48 65522 65533 4
high = 0 65528 5 53 65436 53 5 65528 0 0
low reconstruction = 0 65528 65531 53 100 53 65531 65528 0 0
high reconstruction = 0 65532 65533 14 48 65427 48 14 65533 65532

APPENDIX B - 3D-II ARCHITECTURE'S SCHEDULE

This is the schedule for a 4x4x2 transform architecture.

L1 = 4 L2 = 4 L3 = 2
t = Control
lso = low store offset (for local RAM) = 32
hso = high store offset (for local RAM) = 48

t	GetRAM	Coeff	StoreRAM	
0	0..3	L1	lso+0, hso+0	// L, H
1	4..7	L1	lso+1, hso+1	// L, H
2	8..11	L1	lso+2, hso+2	// L, H
3	12..15	L1	lso+3, hso+3	// L, H
4	16..19	L1	lso+4, hso+4	// L, H
5	20..23	L1	lso+5, hso+5	// L, H
6	24..27	L1	lso+6, hso+6	// L, H
7	28..31	L1	lso+7, hso+7	// L, H
8	lso+0..lso+3	L2	lso+8, hso+8	// LL, LH
9	lso+4..lso+7	L2	lso+9, hso+9	// LL, LH
10	hso+0..hso+3	L2	lso+10, hso+10	// HL, HH
11	hso+4..hso+7	L2	lso+11, hso+11	// HL, HH
12	lso+8..lso+9	L3	lso+12, hso+12	* // LLL, LLH
13	hso+8..hso+9	L3	lso+13, hso+13	* // LHL, LHH
14	lso+10..lso+11	L3	lso+14, hso+14	* // HLL, HLH
15	hso+10..hso+11	L3	lso+15, hso+15	* // HHL, HHH

* The last 4 calculations are outputs, and do not need to be stored in RAM.

Note: The last 4 calculations are done on 2 inputs instead of 4. Because of this, the 2 inputs can be stored next to two 0s so that the input data width remains the same. The two 0s do not have to be explicitly done, since the first two wavelet coefficients are zero.

So lines 12-15 can instead be:

12	lso+6..lso+9	L3	lso+12, hso+12	* // LLL, LLH
13	hso+6..hso+9	L3	lso+13, hso+13	* // LHL, LHH
14	lso+8..lso+11	L3	lso+14, hso+14	* // HLL, HLH
15	hso+8..hso+11	L3	lso+15, hso+15	* // HHL, HHH

This is the schedule for a 4x4x4 transform architecture.

L1 = 4 L2 = 4 L3 = 4
t = Control
lso = low store offset (for local RAM) = 64
hso = high store offset (for local RAM) = 64+28

t	GetRAM	Coeff	StoreRAM	
0	0..3	L1	lso+0, hso+0	// L, H
1	4..7	L1	lso+1, hso+1	// L, H
2	8..11	L1	lso+2, hso+2	// L, H
3	12..15	L1	lso+3, hso+3	// L, H
4	16..19	L1	lso+4, hso+4	// L, H
5	20..23	L1	lso+5, hso+5	// L, H
6	24..27	L1	lso+6, hso+6	// L, H
7	28..31	L1	lso+7, hso+7	// L, H
8	32..35	L1	lso+8, hso+8	// L, H
9	36..39	L1	lso+9, hso+9	// L, H
10	40..43	L1	lso+10, hso+10	// L, H
11	44..47	L1	lso+11, hso+11	// L, H
12	48..51	L1	lso+12, hso+12	// L, H
13	52..55	L1	lso+13, hso+13	// L, H
14	56..59	L1	lso+14, hso+14	// L, H
15	60..63	L1	lso+15, hso+15	// L, H
16	lso+0..lso+3	L2	lso+16, hso+16	// LL, LH
17	lso+4..lso+7	L2	lso+17, hso+17	// LL, LH
18	lso+8..lso+11	L2	lso+18, hso+18	// LL, LH
19	lso+12..lso+15	L2	lso+19, hso+19	// LL, LH
20	hso+0..hso+3	L2	lso+20, hso+20	// HL, HH
21	hso+4..hso+7	L2	lso+21, hso+21	// HL, HH
22	hso+8..hso+11	L2	lso+22, hso+22	// HL, HH
23	hso+12..hso+15	L2	lso+23, hso+23	// HL, HH
24	lso+16..lso+19	L3	lso+24, hso+24	* // LLL, LLH
25	hso+16..hso+19	L3	lso+25, hso+25	* // LHL, LHH
26	lso+20..lso+23	L3	lso+26, hso+26	* // HLL, HLH
27	hso+20..hso+23	L3	lso+27, hso+27	* // HHL, HHH

* The last 4 calculations are outputs, and do not need to be stored in RAM.

ABSTRACT

Weeks, Michael Clark. B.E.S., University of Louisville, 1993; Master of Engineering, University of Louisville, 1994; Master of Science, University of Southwestern Louisiana, 1996; Doctor of Philosophy, University of Southwestern Louisiana, 1998

Major: Computer Engineering

Title of Dissertation: Architectures for the 3-D Discrete Wavelet Transform

Dissertation director: Dr. Magdy Bayoumi

Pages in dissertation: 151; Words in abstract: 350

Volumes of 3-D data, such as Magnetic Resonance Imaging (MRI), desperately need compression. The 3-D Discrete Wavelet Transform (DWT) suits this application well, to give it maximum compression without blocking artifacts. This dissertation compares VLSI architectures for the Discrete Wavelet Transform (DWT) and presents 2 new architectures.

Many architectures for the Discrete Wavelet Transform (DWT) have been proposed since 1990. The DWT is not a straight-forward problem to implement on a chip, so there are several types of designs. The types of architectures depend on the application data's dimensions, and the algorithm used, as well as whether the design is systolic, semi-systolic, folded, or digit-serial. The advantages and disadvantages of each design will be presented in this work.

Medical applications are the targets for the 3-D Discrete Wavelet Transformers. MRI studies generate multiple megabytes of data, and MRIs are of reasonable X and Y dimensions, and have grey-scale values for pixels, so the data precision (8 bits) is also low. Television applications may also be on the horizon for this technology.

The first architecture is an implementation of the 3-D DWT similar to 1-D and 2-D designs. It allows even distribution of the processing load onto 3 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, since the data are operated on in a row-column-slice fashion. The design is cascadable. Due to pipelining, all filters are utilized 100% of the time, except for the start up and wind-down times.

The second architecture uses block inputs to reduce the amount of on-chip memory. It

has a control unit to select which coefficients to pass on to the low and high pass filters. The memory on the chip will be small compared to the input size, since it depends solely on the filter sizes. The filters are parallel, since the systolic filters assume that the data is fed in a non-block form such that partial calculations are done.

These 2 new architectures are some of the first 3-D DWT architectures.

VITA

Michael Clark Weeks was born in Louisville, Kentucky, on May 1, 1968. He is son of R. Clark Weeks and Milly Carricato Weeks. He grew up in Louisville, and attended Saint Xavier High School, graduating in 1986. He went on to study at the University of Louisville Speed Scientific School, in the Engineering Math and Computer Science department. He entered the University of Louisville graduate program in May 1992, and received a Master of Engineering degree in May 1994. After working at IBM Global Services for 7 months, he enrolled in the Computer Engineering department at the University of Southwestern Louisiana's Center for Advanced Computer Studies. Here he received a Master of Science degree in December 1996, and plans to complete the Ph.D. degree in May 1998.

In 1991, he entered Engineers' Days with a game program written in the language C. This project won second place. In 1992, he won first place in Engineers' Days with a beer-brewing program that used a Genetic Algorithm. The following January 1993, he was granted a Graduate Service Assistantship with the Disability Resource Center at the University of Louisville. He was awarded the Outstanding Senior Award by the Speed School Alumni Foundation in May of 1993. At the University of Southwestern Louisiana, he was awarded the Board of Regents Fellowship. Also, he was recognized at the 1996 University of Southwestern Louisiana honors ceremony.