ABSTRACT

JOSEPH, SMITHA Implementation of the Two Dimensional Integer Wavelet Transform for Transmission of Images. (Under the guidance of Dr Winser E. Alexander).

Applications like telemedicine require the diagnostic information in the form of very large images to be transferred using existing networks. These applications use region of interest coding in medical images to reduce the cost while meeting the diagnostic quality requirements. The Integer Wavelet Transform is the major component in the JPEG 2000 based dynamic region of interest coding scheme. The wavelet transform helps to concentrate the signal energy to fewer coefficients to increase the degree of compression when the data is encoded. The wavelet transform is a computationally intensive component and the computations can be accelerated for real time applications like telemedicine by implementing the algorithm in hardware. Since the reconstruction of the medical images should be done with little or no loss to maintain diagnostic accuracy, we are interested in a lossless implementation of the wavelet transform.

This thesis investigated the approaches to be used in the hardware implementation of the two dimensional integer wavelet lifting algorithm using the bi-orthogonal (9,7) filter. It proposes an implementation of a hardware model that meets the stringent quality requirements and is suitable for transmission of images for applications like telemedicine. The major design issues considered in the design of the 2D-integer wavelet transform architecture were the round off errors due to the use of fixed point arithemetic to perform the computations, the waiting time between the row and column processing, the memory access time and the implementation of the non-causal lifting structure. The round off errors in fixed point computations were reduced through scaling, sign extension and rounding. The proposed hardware model has a high throughput rate because it does the row and column processing in a single pass which avoids the waiting time between the row and column processing. This is done with substantial savings in memory over the more traditional level based designs. An architecture based on the data dependency graph was developed that overcame the difficulty posed by the non-causal lifting structure. The results obtained from the test cases on 8 bit gray scale images show that this 2D hardware model yields very good performance. The hardware model reconstructed the test images without any error.

Implementation of the Two Dimensional Integer Wavelet Transform for Transmission of Images

by

Smitha Joseph

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Electrical and Computer Engineering

Raleigh

2004

Approved By:

Dr. Wesley E. Snyder

Dr. W. Rhett Davis

Dr. Winser E. Alexander
Chair of Advisory Committee

I dedicate this thesis to my parents and my teachers.

Biography

Smitha Joseph was born in Kerala, India on 6th February 1979. She received her Bachelor's degree (B.Tech) in Electronics and Telecommunication Engineering from the College of Engineering Thiruvananthapuram (C.E.T), University of Kerala in 2000. During her undergraduate studies, she worked on a project to develop software for image compression using wavelets in the Digital Signal Processing Division of the Indian Space Research Organization (I.S.R.O), Thiruvananthapuram. After graduation, she worked in Wipro Technologies, Bangalore from July 2000 to December 2001 as a Systems Engineer in their Nortel Networks services account.

She joined the Master's program in Electrical Engineering at San Jose State University in Fall 2002 and transferred her graduate studies to North Carolina State University in Spring 2003.

Acknowledgements

I feel lucky to be one of those few people who have had a chance to turn their dream to reality. Even though I have put in lot of effort to achieve my dreams, there are several people in my life who have helped me through with their words and deeds of encouragement and support.

First, I would like to thank my advisor, Dr. Winser E. Alexander, for giving me a research topic in the field in which I have always had a great interest. I appreciate his constant guidance and support in my studies and my research. During the course of this work, I have benefitted greatly from his experience and knowledge. I am grateful to Dr. Wesley E. Snyder and Dr. W. Rhett Davis for agreeing to be on my committee.

I would like to thank the Electrical and Computer Engineering Department at North Carolina State University for giving me financial assistance for studies through a teaching assistantship for 3 semesters.

I would like to thank my parents, Joseph and Gracy who have always encouraged me to set higher goals in life and to work for them. I would like to thank my uncle, Dr Joseph Sebastian for encouraging me to pursue my dreams even if they seemed a little afar. I would like to thank him, his wife Teresa and their children Elizabeth and Kathryn for providing me their support, love and a comfortable, warm stay that I never felt faraway from my home.

I am thankful to my innumerable friends at NC State, in California and in India, all of who provided me with moral support and encouragement whenever I felt frustrated. I would like to express a special thanks to the members of HIPER DSP group who have given me valuable suggestions and support during this work.

I would like to thank my husband George for his love, immense support and understanding. Although we were separated by miles during my Master's studies, he is close to my heart and we will soon be together again.

At this point, I remember an old poem about everyone in this world being weary travelers through life to different destinations and how we meet each other on the roads and at the inns during our journey. I thank God for bringing to my path so many people who have been beacons of light helping me to reach my dreams and my destinations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The latter part of the 20th century brought about a revolution in the way the world communicated. Telemedicine uses the technological advances in telecommunications to the advantage of health professionals. It also helps to provide better health care services to the consumers. Telemedicine requires the transfer of diagnostic information such as X-rays, computed radiography etc using the existing networking infrastructure. Even though direct availability of medical images in digitized form is not a problem, the large amount of data present and the stringent quality requirements pose a major bottle-neck in the widespread use of telemedicine. Hence, this necessitates research to develop new algorithms or to improve the existing algorithms to meet these requirements.

Several methods to achieve lossy and lossless image compressions have been proposed. Lossy techniques are not acceptable in telemedicine as they might lead to errors in diagnosis. Lossless techniques help in accurate diagnosis but result in higher costs due to their need for higher bit rates. Hence, an optimum solution should provide lossless quality in the diagnostically important regions while encoding other portions of the image using lossy compression.

We must know the fundamentals of image coding to study the region of interest coding in images. The JPEG algorithm is one of the most widely used standards in image

compression. JPEG 2000 is the latest image compression standard developed by the JPEG committee. This chapter begins by examining the different stages in the JPEG 2000 codec, and then uses these concepts to establish a general framework for the study of the region of interest coding algorithm used as the basis of this thesis.

1.1.1 JPEG 2000 Standard

JPEG 2000 offers several new features and improvements over the older JPEG standard. Besides supporting both lossless and lossy compression, JPEG 2000 offers superior performance even at low bit rates. It has good error resilience properties, thus making it useful in Internet and mobile applications. One of the key features supported by JPEG 2000 is the region of interest coding scheme which allows the regions of interest in the image to be reconstructed at the receiver side with lossless quality.

JPEG 2000 is a wavelet transform based codec and the various stages involved are shown in the block diagram given in figure 1.1.

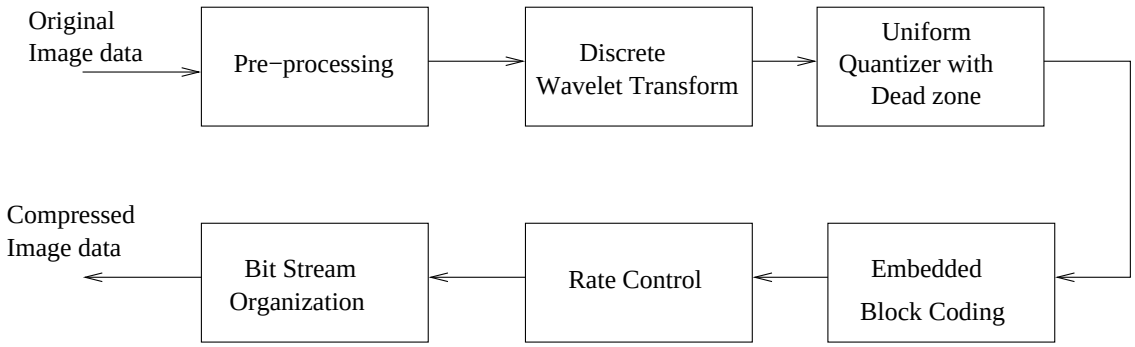

Figure 1.1: Block diagram of JPEG 2000 Encoder

Preprocessing

The Preprocessing stage in the JPEG 2000 Codec consist of three substages as shown in the block diagram of figure 1.2.

If the image to be encoded is larger than the memory available in the encoder, JPEG 2000 allows optional tiling. In the tiling stage, the image is broken down to non

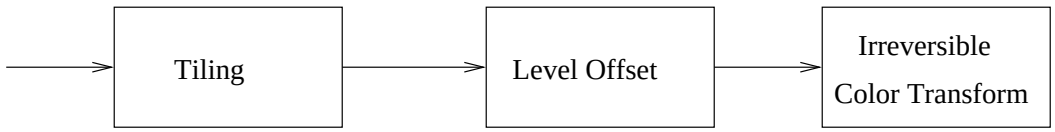

Figure 1.2: Pre-processing substages

overlapping tiles of smaller equal sizes.

Since JPEG 2000 uses high pass filtering, the input data should have a dynamic range around zero. Thus the unsigned image values have to be offset by a dc value to get signed integers centered around zero. For example, in a 8-bit gray scale image, the pixels have values in the range $0 - 255$ (0 *to* $2^8 - 1$). In the pre-processing stage, all the values are offset by -128 (-2^7). Thus after level offset the pixel values are in the range -128 *to* 127.

Color images are commonly represented in the RGB format and consist of 3 component planes, one each in red, green and blue. Each component plane is passed through the wavelet transform and the results are compressed separately. If the color image is represented in terms of it's illuminance(Y) and chrominance components(Cr,Cb), the statistical dependence between adjacent pixels is greater and the image can be compressed better. Therefore in JPEG 2000, the RGB components of a color image are transformed to YCrCb components by using an irreversible color transform.

Wavelet Transform

JPEG2000 uses the wavelet transform to decompose the data in the image into subbands. These subbands can be coded more efficiently than the original data due to their statistical properties. Even though the wavelet transform used by JPEG 2000 is one dimensional (1D-DWT) in nature, we obtain a two dimensional discrete wavelet transform(2D-DWT) by applying the 1D-DWT across the rows and columns of the image as a product separable transform.

1D-DWT Given a sequence $x[n]$, we can obtain the 1D-DWT by filtering and subsampling as depicted in figure 1.3. The input data is initially filtered using an analysis filter bank consisting of a high pass filter, $h[n]$ and low pass filter, $g[n]$. The filtered signals are then

subsampled by a factor of 2 to give $s0[n]$ and $d0[n]$, the low pass and high pass subband sequences respectively.

$$s0[n] = \sum_{k=-\infty}^{\infty} x[k]h[2n-k] \tag{1.1}$$

$$d0[n] = \sum_{k=-\infty}^{\infty} x[k]g[2n-k] \tag{1.2}$$

The subsampling helps to maintain the sample rate constant. Thus the forward discrete wavelet transform (DWT) decomposes a 1D sequence to give two sequences with half the number of the samples in the original sequence.

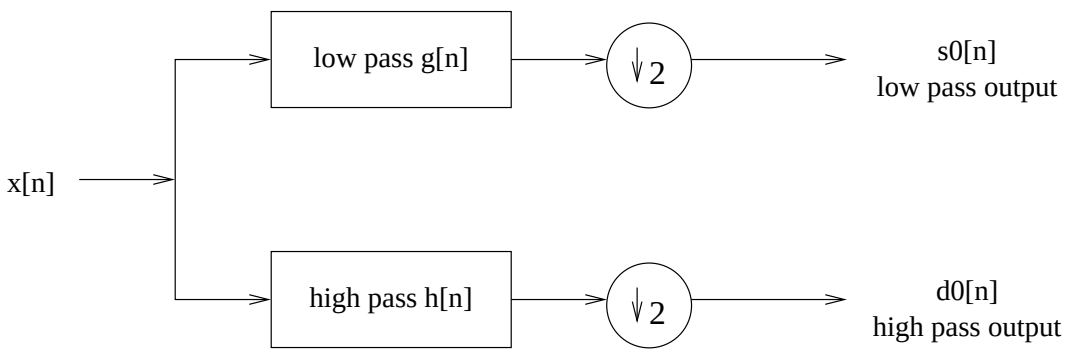

Figure 1.3: 1D-DWT Structure

2D-DWT In JPEG 2000, the number of levels of the 2D-DWT performed is implementation dependent. Figure 1.4 shows the filter bank structure used in each level of the pyramidal decomposition of an image. The filters $h[n]$ and $g[n]$ are one dimensional filters.

We get four subbands (LL,LH,HL,HH) from one level of the 2D-DWT. The LL subband contains the low level details of the image. In the next level, the 2D-DWT of the LL subband is obtained and this is repeated in each succeeding level.

Quantization

Quantization is a process to reduce the precision of the coefficients, thus helping to obtain higher compression ratios. All the coefficients obtained after applying the wavelet

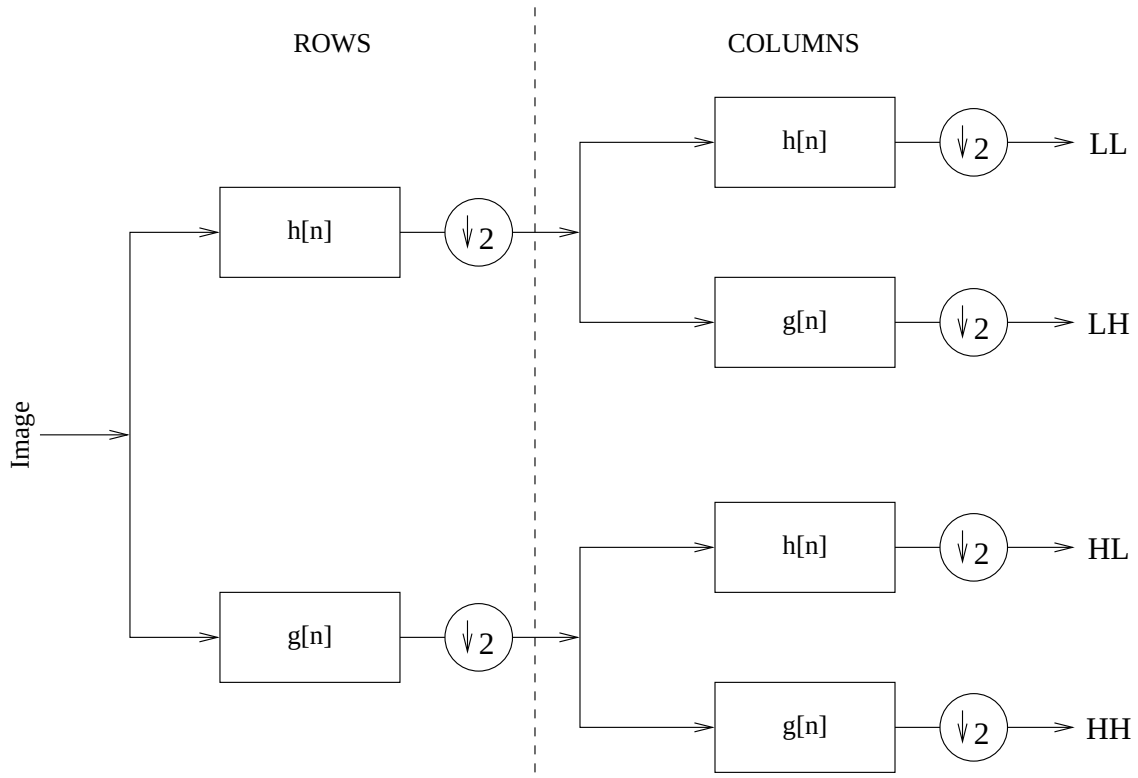

Figure 1.4: Filter bank structure used in the wavelet decomposition of an image

transform are quantized using a uniform quantizer with dead zone. In each subband b, all the coefficients are quantized using a basic step size δ_b. The quantizer step size for a particular subband is chosen depending on the significance of the subband in the final reconstruction of the image. After choosing the quantizer step size, the quantization rule given in equation 1.3 is applied to each coefficient in that subband

$$Q(x) = sign(x)\lfloor |x|/\delta_b \rfloor \tag{1.3}$$

where x is the input coefficient, δ_b is the quantizer step size, $|x|$ is the absolute value of x, $sign(x)$ is the sign of the coefficient x, $\lfloor x \rfloor$ represents the largest integer less than x and $Q(x)$ is the output of the quantizer. The dead zone quantizer structure is as shown in figure 1.5.

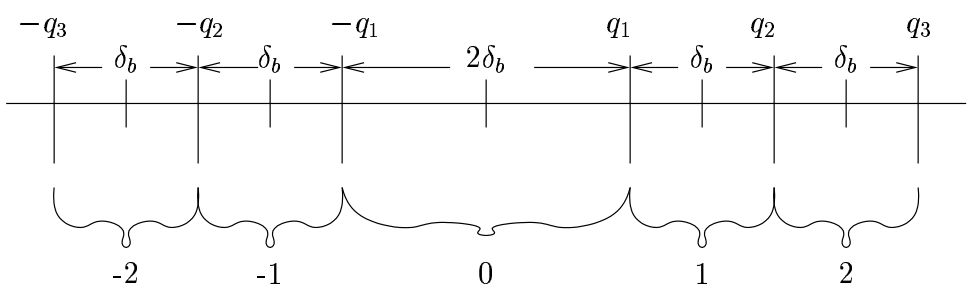

Figure 1.5: Uniform quantizer with dead zone

The quantization range around 0 is $2\delta_b$ which is called the deadzone. For example, let $x = -32.64$ and quantizer step size be 10, then $Q(x) = -\lfloor 32.64/10 \rfloor = -3$. Any value of x between -10 and +10 falls in the dead zone giving the quantizer output 0.

Embedded Block Coding

One of the most important properties of the wavelet transform is that the energy present in the input signal is usually compacted in a very small number of wavelet coefficients. This compaction of energy can bring about a large degree of compression if we code only the high energy coefficients. The disadvantage of this method is that the position of the coefficient also needs to be sent for proper decoding and the position information will

utilize a significant portion of the bit rate. JPEG 2000 uses a method known as embedded block coding to lower the cost of encoding the significant coefficient position [17].

In JPEG 2000, each quantized subband is broken down into non overlapping rectangles of equal size with height and width equal to the powers of 2 (eg: 64 X 64). These non overlapping rectangles are called code blocks. Each code block is then encoded separately using the embedded block coding algorithm explained in [10]. In this algorithm, each bit plane in a code block is encoded in three sub bit plane passes instead of single pass. The set of three passes generates an embedded code for each bit plane of the code block and is compressed using a context dependent binary arithmetic encoder. Thus embedded codes generated by several code blocks for each bit plane form the quality layers. Each additional layer improves the quality of the image reconstructed at the receiver side.

Decoding

The JPEG 2000 decoder does the opposite of the actions done by the encoder, giving the recontructed image with losses or without losses depending on the encoding scheme followed. The codestream at the receiver side is passed through the arithmetic decoder, inverse quantizer, inverse wavelet transform and inverse color transform blocks. The reconstructed image may not be lossless. The losses are introduced by the quantization of the wavelet coefficients.

1.1.2 Region of Interest Coding in JPEG 2000

JPEG 2000 allows certain regions of the image, which are of interest to the receiver to be coded with greater quality than the background image. This is especially useful in applications like telemedicine, satellite imagery etc. There are two types of Region of Interest (ROI) coding. In both types of coding, the wavelet coefficients corresponding to the ROI must be identified and coded to have better quality than the background coefficients.

Static ROI The regions of interest are defined by the user at the time of encoding. The wavelet coefficients corresponding to this region are identified by using a ROI mask. All the coefficients contributing to the inverse wavelet transform of the ROI have to be encoded without losses to ensure higher quality for the ROI in the reconstructed image.

Figure 1.6 demonstrates lossless ROI mask generation.

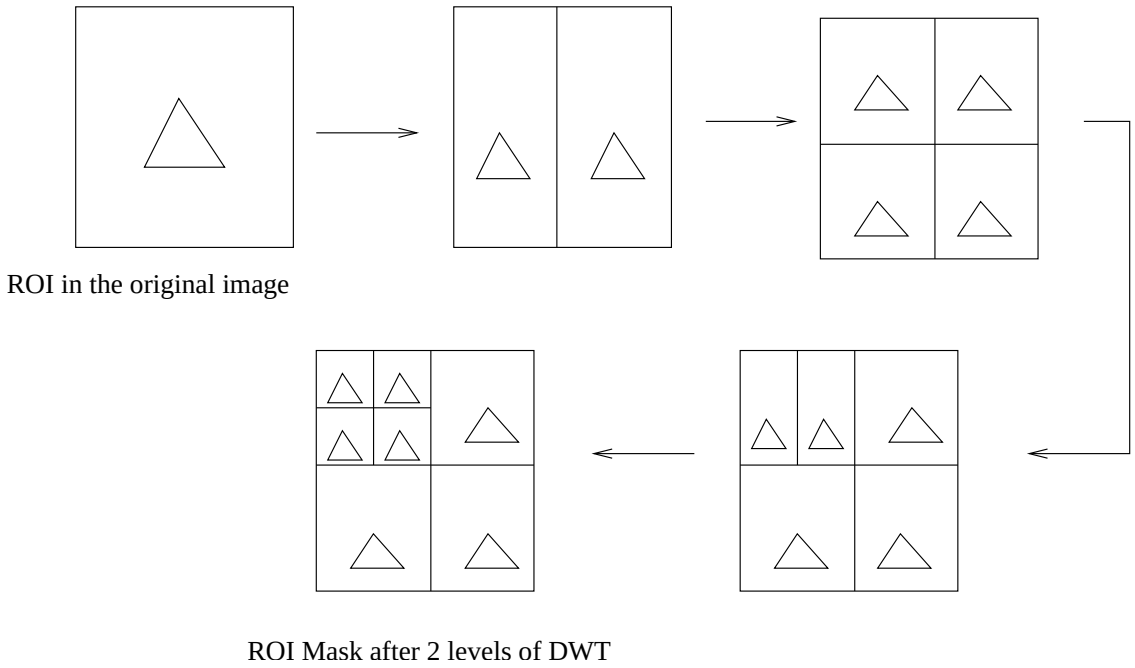

Figure 1.6: ROI mask generation

These ROI coefficients are scaled compared to the coefficients for the background. Thus the bits associated with these ROI coefficients are placed in higher bit planes. The bit planes are transmitted or stored from the highest to the lowest. Hence during the decoding at the receiver, the ROI is decoded before its background in resolution and quality. The two methods of scaling suggested are MAXSHIFT scaling and general scaling.

In MAXSHIFT scaling, the ROI coefficients are scaled up by s bit planes such that the least significant bit for the ROI coefficients is higher than the most significant bit of the background (BG) coefficients. The decoder can scale all BG coefficients whose value will be less than the s threshold by s bit planes. Thus the decoder does not require the ROI mask information.

In general scaling, the wavelet coefficents identified by the ROI mask are shifted upwards by s bit planes. The value of s is encoded in the bit stream and is used by the decoder to shift down the coefficients after decoding. The BG coefficients remain unchanged. The ROI mask is needed both at the encoder and at the decoder. For more details of scaling

methods refer to [10], [14].

Dynamic ROI The regions of interest are pointed out by the client during the interaction with the server. The server dynamically generates and sends additional layers to give better quality in the regions of interest. As presented by [8], one method is to quantize the wavelet coefficients generated and encode the corresponding bit planes without any ROI specifications. When the ROI is requested at any point during transmission, a ROI marker indicating the ROI index and mask is inserted in the bit stream. Initially, only a part of the bit planes in each subband are sent. After the ROI is requested, all the bit planes for the coefficients in the ROI are arithmetically encoded and sent.

1.2 ROI Coding in Medical Image Compression

ROI coding for use in medical image compression was presented by Sung Yoon [20]. In this proposed scheme, a post segmentation approach was used. The region scalable coding was achieved by using the Discrete Wavelet Transform and successive quantization as used in JPEG 2000. The transformed and quantized data was partitioned into sets using the method given in [15]. A set thus consisted of a pixel in the lowest subband and children from the quad tree structure as shown in figure 1.7.

The wavelet coefficients in each set were scanned in multiple passes using successively decreasing thresholds. Each pass generated a bit plane and this method of successive quantization is similar to the method described in [15]. Each bit plane acted as a layer for the corresponding set. Figure 1.8 illustrates an example of generating bit planes from a given set. The example shows only three quantization steps, the process was continued until all of the bit planes were generated.

The bit planes generated for each set were compressed by the use of run length encoding. Three mark up symbols were used in coding - the head coefficient of tree (HOT), the end of block code (EOB) and the end of set code (EOS). The header containing information regarding markers, number of iterations and number of bits in each set was sent before the sets were sent. Thus any sets corresponding to the ROI could be extracted and transmitted due to the efficient structure of the encoded data.

Sung Yoon showed that among the three representative types of filters Haar, bi-

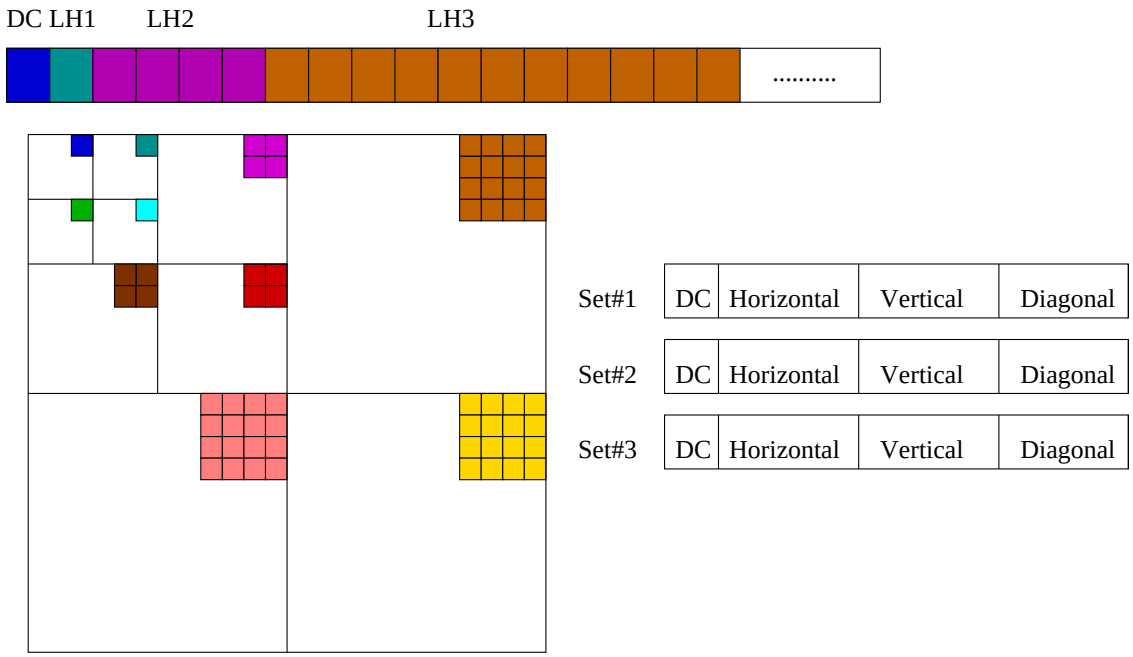

Figure 1.7: Set partitioning of the wavelet transformed data

orthogonal (5,3) and bi-orthogonal (9,7) used in his studies of ROI coding, the (9,7) linear phase filter offered the best performance. The ROI was defined in the spatial domain, while quantization and layering was performed in the wavelet domain. Hence the filter length affected the areas around the boundary of the ROI, which in turn degraded the quality of the picture in the ROI. The (9,7) linear phase filter has longer length of support than the other two filters and thus gave better performance.

1.3 Objective of the Thesis

The ROI coding algorithm discussed in the previous section is computationally intensive. The algorithm can be implemented in hardware to accelerate the computations for real time applications such as telemedicine. The scope of this thesis was limited to the lossless implementation of the wavelet transform which is a part of the ROI scheme proposed by Sung Yoon [20]. The core part of the ROI coding scheme is the two dimensional Integer Wavelet Transform(IWT). The reconstruction of the image after integer wavelet

	63	34	49 10 14 −13	Set #k
Quantization step #1				
Threshold T0=32	1	1	1 0 0 0	Bitplane #1
	31	3	17 10 14 −13	Subset #k after Quantization step #1
Quantization step #2				
Threshold T0=16	1	0	1 0 0 0	Bitplane #2
	15	3	1 10 14 −13	Subset #k after Quantization step #2
Quantization step #3				
Threshold T0=8	1	0	1 1 1 − 1	Bitplane #3
	7	3	1 2 6 −5	Subset #k after Quantization step #3

Figure 1.8: Example to show bit plane generation from subset

decomposition has to be as near perfect as possible to meet the stringent quality requirements in telemedicine. This thesis investigated the approaches to be used for the hardware implementation of the two dimensional integer wavelet transform. The research goal of this thesis was to design a hardware implementation of one level of the 2D-IWT using the (9,7) bi-orthogonal filter as used in JPEG 2000. The Verilog hardware description language was selected to capture and model the hardware at the register transfer level(RTL). The implementation was desired to have high throughput while minimizing the error in reconstruction. The hardware model designed was tested using 8 bit gray scale images to verify the functioning of the model and to measure the performance.

Chapter 2

Wavelet Transforms in JPEG2000

2.1 Introduction

The Discrete Wavelet Transform (DWT) is a widely used multiresolution analysis tool. The time frequency characteristics of the wavelet transform allows it to capture the signal energy within a few transform coefficients. This property makes the DWT an useful tool in signal analysis, signal compression etc [9], [16]. In this thesis, we discuss the implementation of the DWT for decomposition and lossless reconstruction of medical images using the inverse DWT.

This chapter gives a brief introduction to the filtering methods and the filter structures used in the DWT. Then the motivation for the use of the lifting structure and the lifting structure for the wavelet filters used in JPEG 2000 is discussed.

2.2 Wavelet Transform

Multiresolution representations are useful in analyzing the information content of a signal. It helps to analyze the different frequencies present in the signal at different resolutions. The wavelet transform of a signal provides the time frequency representation of the signal without changing the information content of the signal.

The wave is an infinite length continous function in time or space, whereas the

wavelet is a localized wave which has its energy concentrated in time or space. The basic idea of the wavelet transform is to represent the signal to be analyzed as a superposition of wavelets. Thus wavelet analysis is similar to Short Time Fourier Transform (STFT) analysis. The signal when transformed using the STFT is multiplied with a window function of constant width and the Fourier transform of each segment is computed. In the wavelet transform, the signal is multiplied by a wavelet function and the transform for each segment of the signal is computed. Since the wavelet function changes its width with each spectral component, the wavelet transform overcomes the inability of the STFT to give good time resolution at higher frequencies and to give good frequency resolution at lower frequencies.

2.2.1 Continous Wavelet Transform

The continous wavelet transform (CWT) of a signal $x(t)$ is defined as given in equation 2.1

$$X_{WT}(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi^*(\frac{t-\tau}{s}) \tag{2.1}$$

The resultant transform, $X_{WT}(\tau, s)$ is a function of two variables, translation, τ and scale, s. In equation 2.1, $\psi(t)$ is the mother wavelet or the basis function and it is used as the prototype for generating all the basis window functions. The translation parameter, τ gives the time information in the wavelet transform. It indicates the location of the window, as it is shifted through the signal. The scale parameter, s defined as $\frac{1}{frequency}$ gives the frequency information in the wavelet transform. Scaling results in the compression or dilation of the window. Thus the low scales, corresponding to wavelets or windows of smaller width give the detailed information in the signal. The high scales, corresponding to wavelets or windows of larger width give a global view of the signal.

The computation of the CWT can be made easier by changing the integral to sum, that is by discretizing the transform into a wavelet series. The discretization is done by sampling the time frequency plane. The Nyquist sampling criterion should be followed to allow perfect reconstruction of the signal during synthesis. The sampling rate can be reduced at higher scales (lower frequencies) in accordance with the Nyquist criterion resulting in fewer computations.

2.2.2 Discrete Wavelet Transform

The computation of the wavelet series requires significant computational time and resources. It is possible to reduce this by using a subband coding algorithm which yields a faster wavelet transform. The wavelet transform of a signal using the CWT is obtained by changing the scale of the analysis window, shifting the window in time, multiplying the signal and integrating the result over all time. In the case of the DWT, the wavelet transform is obtained by filtering the signal through a series of digital filters at different scales. The scaling operation is done by changing the resolution of the signal by subsampling.

The DWT can be computed using either the convolution based or the lifting based procedures [5]. In both methods, the input sequence is decomposed into low pass and high pass subbands, each consisting of half the number of the samples in the original sequence [14].

2.2.3 Convolution Based

The input is filtered using a filter bank consisting of low pass and high pass filters and downsampled by a factor of 2 in the convolution based DWT computational procedure. Figure 1.3 shows one level of the DWT. This can also be computed mathematically using equations 2.2 and 2.3

$$s[n] = \sum_{k=-\infty}^{\infty} x[k]h[2n-k] \tag{2.2}$$

$$d[n] = \sum_{k=-\infty}^{\infty} x[k]g[2n-k] \tag{2.3}$$

where s is the low pass signal and d is the high pass signal.

2.2.4 Lifting Based

The lifting scheme is a much more efficient method to calculate the wavelet transforms than the classical convolution method. The original motivation for the development of the lifting technique was the implementation of second generation wavelets. Second generation wavelets unlike the first generation wavelets do not use the translation and dilation of the same wavelet prototype in different levels. The lifting scheme is a general scheme

and is not limited to developing a filter structure for second generation wavelets. It can also be used to build a ladder type structure for first generation wavelet filters. Any classical wavelet filter bank can be decomposed into lifting steps through the use of the Euclidean algorithm which will be shown later in this chapter.

The cost of computations in the lifting scheme is half that of the standard convolution scheme for long FIR filters [7]. The lifting scheme also offers several advantages such as in-place computation of the DWT, integer to integer wavelet transform, symmetric inverse and forward transform etc [3]. The lifting scheme simplifies the DWT computation by looking at the basic principles behind the use of wavelets. The wavelet transform of a one dimensional signal gives a multiresolution representation of the signal. At each resolution level, the wavelet basis functions decorrelate the information in the signal and we get a high pass and a low pass component. Thus the basic idea of the wavelet transform is to build a sparse approximation exploiting the correlation present in the input signal. The lifting scheme is a technique which helps to yield this smaller approximation with fewer computations.

Predict and Update

There are two different ways to introduce the lifting scheme. One approach is to discuss how lifting affects the wavelet transform [5]. The other approach is to show how the lifting steps can be derived from the filter bank structure used for the DWT computation. In this section, we show the first approach of how the lifting scheme is done and how it resembles the wavelet transform representation. The second approach will be explained in the next section.

The lifting scheme consists of three stages: split, predict and update. Consider a one dimensional sequence or signal $x[n]$ whose elements have some correlation between them. We can represent the signal compactly by exploiting the correlation structure present in it [5].

The first step in the lifting scheme is splitting the input signal $x[n]$ into two subsets, the even sample set ($s_0[n]$) and the odd sample set ($d_0[n]$). This splitting is called the Lazy wavelet transform. Even though the splitting into subsets can be done in many ways, for the ease of reconstruction the Lazy wavelet transform is used.

Since we would like to get a more compact representation of the original sequence

x[n], the aim is to get a sparser approximation of one of the subsets. This done by using the predict step where the linear combination of elements in one subsequence is used to predict the values of the other subsequence using the assumption that the two subsequences produced in the splitting step are correlated. If the correlation present in the original data is high, the predicted values will be close to the actual values.

$$d_1[n] = d_0[n] - \sum_k p[k]s_0[n-k] \tag{2.4}$$

The predict step is shown in equation 2.4, where $p[k]$ is the prediction coefficient. The linear combination of the even subsequence values is used to predict the odd subsequence values. The detail variable (d_1) records the difference between the actual value and the predicted value. If the original sequence is smooth, the detail sequence will be a sparse set. After the predict step, the original sequence is represented in terms of the even samples, $s_0[n]$ and the detail values, $d_1[n]$. The detail sequence can be considered to be equivalent to the sequence obtained after subsampling the output from the high pass wavelet analysis filter.

The predict step results in the loss of some basic properties of the signal like the mean value, which needs to be preserved. The update step lifts the even sequence values using the linear combination of the predicted odd sequence values so that the basic properties of the original sequence is preserved [5]. The even sequence values s_1 obtained as the result of equation 2.5 is equivalent to the subsampled low pass version of the original sequence.

$$s_1[n] = s_0[n] - \sum_k u[k]d_1[n-k] \tag{2.5}$$

The DWT of a one dimensional signal using a simple lifting scheme with one pair of lifting substeps will follow the steps summarized as below

1. Split Step: The input signal $x[n]$ is split into odd and even subsequences, $d[n]$ and $s[n]$ respectively (also known as the Lazy Wavelet Transform)

2. Predict Step: This step predicts data in the subsequence $d[n]$ using the samples in $s[n]$ and replaces the samples in $d[n]$ using the prediction error.

   $d[n] \leftarrow d[n] - P(s[n])$

3. Update Step: This step updates the data in $s[n]$ using the data in $d[n]$.

   $s[n] \leftarrow s[n] + U(d[n])$

Filter Banks using Euclidean algorithm

The previous section introduced the lifting scheme through the implementation of a second generation wavelet. We can use this powerful scheme to decompose every FIR wavelet or filter bank into lifting steps. The filter bank structure used in the DWT computation at each level can thus be replaced by this more efficient computational scheme to give a low resolution part and a high resolution part.

If we filter a signal $x[n]$ using a FIR filter with impulse response h[n], the output of the filter is written as

$$Y(z) = H(z)X(z) \tag{2.6}$$

If we downsample the output of the filter keeping only the even samples, we get

$$Y_e(z^2) = \frac{(Y(z) + Y(-z))}{2} \tag{2.7}$$

Similarly, downsampling the output of the filter keeping only the odd samples, we get

$$Y_o(z^2) = \frac{(Y(z) - Y(-z))}{2z^{-1}} \tag{2.8}$$

Thus we can write $Y(z)$ as

$$Y(z) = Y_e(z^2) + z^{-1}Y_o(z^2) \tag{2.9}$$

The filtering operations in the 1D-DWT decomposition shown by the figure 1.3 is written as

$$\begin{bmatrix} LP(z) \\ HP(z) \end{bmatrix} = \begin{bmatrix} H(z) \\ G(z) \end{bmatrix} X(z) \tag{2.10}$$

After downsampling the filter outputs and taking only the even samples, we get

$$LP_e(z^2) = \frac{(LP(z) + LP(-z))}{2} \tag{2.11}$$

$$HP_e(z^2) = \frac{(HP(z) + HP(-z))}{2} \tag{2.12}$$

In the matrix form, this is written as

$$\begin{bmatrix} s(z) \\ d(z) \end{bmatrix} = \begin{bmatrix} LP_e(z^2) \\ HP_e(z^2) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} H(-z) & H(z) \\ G(-z) & G(z) \end{bmatrix} \begin{bmatrix} X(-z) \\ X(z) \end{bmatrix} \tag{2.13}$$

By first filtering and then subsampling the outputs, we are throwing away half the computed values. Hence it would be more efficient if we subsample before we filter the signal. Then we will be calculating only the even parts of $LP(z)$ and $HP(z)$.

$$LP_e(z) = [H(z)X(z)]_e \tag{2.14}$$

$$= [H_e(z)X_e(z) + z^{-1}H_o(z)X_o(z)] \tag{2.15}$$

$$HP_e(z) = [G(z)X(z)]_e \tag{2.16}$$

$$= [G_e(z)X_e(z) + z^{-1}G_o(z)X_o(z)] \tag{2.17}$$

Thus we can write

$$\begin{bmatrix} s(z) \\ d(z) \end{bmatrix} = P(z) \begin{bmatrix} X_e(z) \\ z^{-1}X_o(z) \end{bmatrix} \tag{2.18}$$

where P(z) is the polyphase matrix.

$$P(z) = \begin{bmatrix} H_e(z) & H_o(z) \\ G_e(z) & G_o(z) \end{bmatrix}$$

Daubechies and Swelden showed in the paper [7], that the polyphase matrix can be factorized into a sequence of alternating upper and lower triangular 2 x 2 matrices and a diagonal normalization matrix. Thus every FIR wavelet filter can be obtained by starting with the Lazy wavelet transform followed by n lifting and dual lifting steps as shown in equation 2.19. The number of lifting steps required depends on the length of the filter.

$$P(z) = \prod_{i=1}^{n} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \tag{2.19}$$

where $s_i(z)$ and $t_i(z)$ are the Laurent polynomials and K is the normalization factor.

From this the inverse wavelet transform can be written as

$$P^{-1}(z) = \prod_{i=1}^{n} \begin{bmatrix} 1 & 0 \\ -s_i(z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & -t_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/K & 0 \\ 0 & K \end{bmatrix} \tag{2.20}$$

In order to compare the computational complexity of the lifting scheme with that of standard convolution scheme, we can compare the cost in terms of additions and multiplications required in each method. Consider a low pass filter $h[n]$ of order $2N$ and a high pass filter $g[n]$ of order $2M$. The cost of standard convolution algorithm is $4(N+M)+2$ ($2M$ additions + $2M+1$ multiplications + $2N$ additions + $2N+1$ multiplications). Using the Euclidean algorithm, the cost of the lifting scheme including the normalizing step is $2(N+M+2)$. Thus the computational savings using lifting scheme is about one half that of the convolutional scheme for longer FIR filters [7].

2.3 Reversible Integer Wavelet transforms

The wavelet transform usually produces floating point coefficients even when applied to integer sequences. The original integer data can be reconstructed perfectly in theory by using these coefficients. However in practice, we usually use the fixed point format for data values as fixed point systems are easier to implement. The reduced precision arithmetic used in such systems can introduce round off errors in the computations. Hence in applications where we need lossless reconstruction, we need transforms which have the reversibility property even when reduced precision is used.

It was shown by Calderbank et al., [5], that we can build wavelet transforms that map integers to integers using the lifting structure. The reversibilty property is obtained by rounding off the predict filter or update filter output before adding or subtracting in each lifting step. Hence the lifting steps at level i decomposition become

$$d_{i,1}[n] = d_{i,0}[n] - \lfloor \sum_k p_i[k] s_{i,0}[n-k] + 1/2 \rfloor \tag{2.21}$$

$$s_{i,1}[n] = s_{i,0}[n] - \lfloor \sum_k u_i[k] d_{i,1}[n-k] + 1/2 \rfloor \tag{2.22}$$

These lifting steps are invertible and the inverse lifting steps can be obtained by reversing the operations and flipping the signs.

Even though the lifting step now results in an integer to integer transform, the normalization step using the scaling factor will reintroduce floating point coefficients. This issue can be avoided by using one of the two methods

1. Omitting the scale factor K, and keeping in mind that the actual lowpass value will be obtained by multiplying the lowpass value obtained through lifting by the scale factor and the actual high pass value by dividing the high pass value obtained via lifting with the scale factor. We can obtain coefficients through using only lifting steps by choosing a factorization method that makes K as close to 1 as possible.

2. With three extra lifting steps that make the scale factor equal to 1. By choosing to factorize in this way, we get integer to integer transforms through using lifting steps.

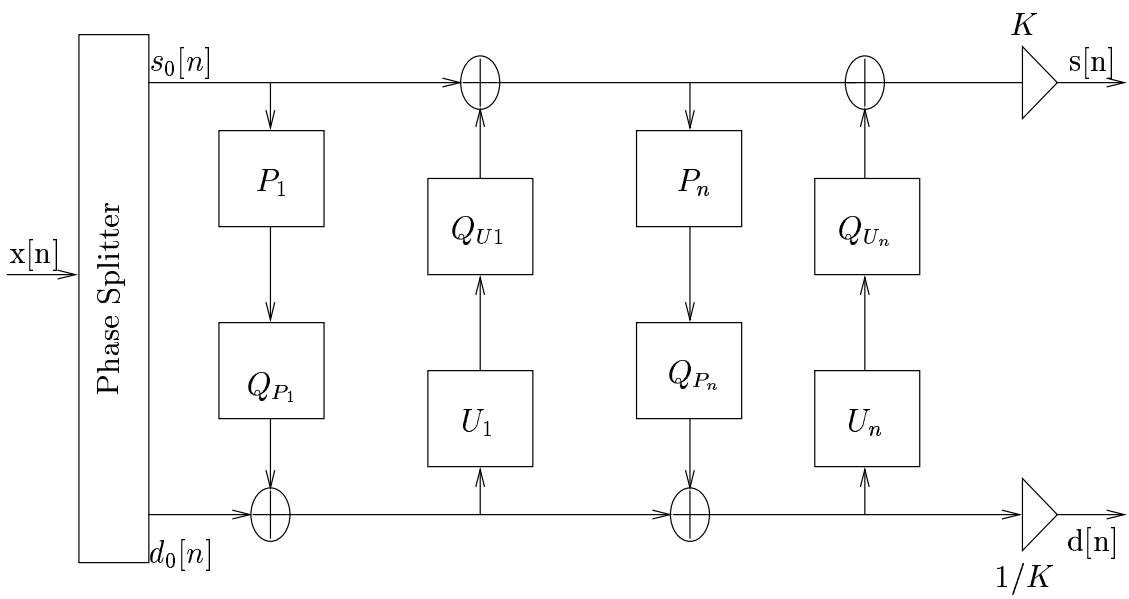

Figure 2.1: Block diagram of forward integer to integer transform

The block diagram in figure 2.1 shows the forward integer to integer transform. The inverse integer to integer transform is obtained by reversing the computations in the forward transform, using appropriate sign changes.

Table 2.1: Two Sets of Linear Phase Biorthogonal Wavelet Filter Coefficients

Filter Index	(9,7) Filter coefficients		(5,3) Filter coefficients	
	h_0	g_0	h_0	g_0
0	0.852699	0.788486	1.060660	0.707107
-1,1	0.377402	0.418092	0.353553	0.353553
-2,2	-0.110624	-0.040689	-0.176777	
-3,3	-0.023849	-0.064539		
-4,4	0.037828			

2.4 Integer Wavelet Filter Examples

The two filters which are supported by JPEG 2000 are the (5,3) Integer filter and the Daubechies (9,7) filter. The table 2.1 shows the coefficients of the biorthogonal linear phase (5,3) and (9,7) wavelet filters.

We will examine the integer to integer transform lifting steps for these two filters. Let the input signal be x[n], low pass signal s[n] and high pass signal d[n] [1]. The lazy wavelet transform step is given by equations 2.23 and 2.24

$$s_0[n] = x[2n] \tag{2.23}$$

$$d_0[n] = x[2n+1] \tag{2.24}$$

The polyphase matrix for the biorthogonal (5,3) filter after normalizing the filter coefficients with $\frac{1}{\sqrt{2}}$ is

$$P(z) = \begin{bmatrix} -\frac{1}{8}z + \frac{6}{8} - \frac{1}{8}z^{-1} & \frac{2}{8} + \frac{2}{8}z \\ -\frac{1}{2} - \frac{1}{2}z^{-1} & 1 \end{bmatrix}$$

The lifting steps for the (5,3) filter given in the equations 2.25 and 2.26 are obtained after factorizing the polyphase matrix.

$$d[n] = d_0[n] - 0.5(s_0[n+1] + s_0[n]) \tag{2.25}$$

$$s[n] = s_0[n] - 0.25(d[n] + d[n-1]) + 0.5) \tag{2.26}$$

The factorization which is symmetric with every quotient a multiple of $(z+1)$ was chosen although many factorizations exist for the polyphase matrix for the (9,7) filter. The factorized polyphase matrix is given by equation 2.27.

$$P(z) = A * B * C * D * \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \tag{2.27}$$

where

$$A = \begin{bmatrix} 1 & \alpha_0(1+z^{-1}) \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ \alpha_1(1+z) & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & \alpha_2(1+z^{-1}) \\ 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 \\ \alpha_3(1+z) & 1 \end{bmatrix}$$

$\alpha_0 = 1.586134, \alpha_1 = 0.052980, \alpha_2 = 0.882911 and \alpha_3 = 0.443506$[1]. The normalization factor for the (9,7) filter, $K = 0.812893$. The lifting steps for the (9,7) bi-orthogonal filter are thus given by the following equations

$$d_1[n] = d_0[n] - \alpha_0 * (s_0[n+1] + s_0[n]) \tag{2.28}$$

$$s_1[n] = s_0[n] - \alpha_1 * (d_1[n] + d_1[n-1]) \tag{2.29}$$

$$d_2[n] = d_1[n] + \alpha_2 * (s_1[n+1] + s_1[n]) \tag{2.30}$$

$$s_2[n] = s_1[n] + \alpha_3 * (d_2[n] + d_2[n-1]) \tag{2.31}$$

$$s[n] = K * s_2[n] \tag{2.32}$$

$$d[n] = 1/K * d_2[n] \tag{2.33}$$

The cost of computation using the convolution scheme for the (9,7) filter to get the low pass wavelet coefficients and the high pass wavelet coefficients is 30 (16 multiplications +14 additions). The cost using the lifting scheme is 14 (6 multiplications +8 additions). Thus the computational cost in lifting is less than half that of the convolution scheme.

Chapter 3

Review of Architectures for the Discrete Wavelet Transform

3.1 Introduction

In recent years, after the emergence of the JPEG 2000 image compression standard, a lot of research has been done to develop an efficient 2D-DWT architecture. This is due to the considerable amount of computation time required by this core part of the compression standard.

The main bottle-neck in the 2D-DWT is the memory access time required for the horizontal and vertical filtering. As explained in section 1.1.1, the 2D-DWT computation consists of two stages. The initial stage computes the one dimensional wavelet transform of the 2D-data rowwise. In the next stage, this rowwise processed data is used for column processing. This chapter reviews the state of the art in 2D-DWT architectures. The DWT architectures can be categorized into three main types - level based, line based and block based architectures [4]. The efficiency of an architecture is measured by the memory needed for the computation of the 2D-DWT and the communications required between the parallel processors.

3.2 Level-based Architecture

In the level based architecture, the same processor usually does the row wise and column wise DWT. In most of the cases, the 2D-data is first read row wise from the external memory, and is transformed by the processor. The result is written back to the external memory. On completion of the row processing, the data is read column wise, processed and the resulting coefficients are written in the external memory. The processing for the next level cannot start until the processing for the current level has been completed [13].

Po-Cheng Wu and Liang-Gee Chen [19] proposed a level by level architecture for 2D-DWT using the convolutional filtering method. The architecture consisted of a RAM module, a transform module, external memory and a multiplexer. During the first level of decomposition, the multiplexer chose the image data stored on the external memory. The transform module decomposed the image data into LL, LH, HL and HH subbands. The LL subband was stored in the RAM module of size $\frac{N^2}{4}$ where the image was of size N x N. After the first level of decomposition, the multiplexer selected the data from the RAM module for further levels. The block diagram is shown in figure 3.1

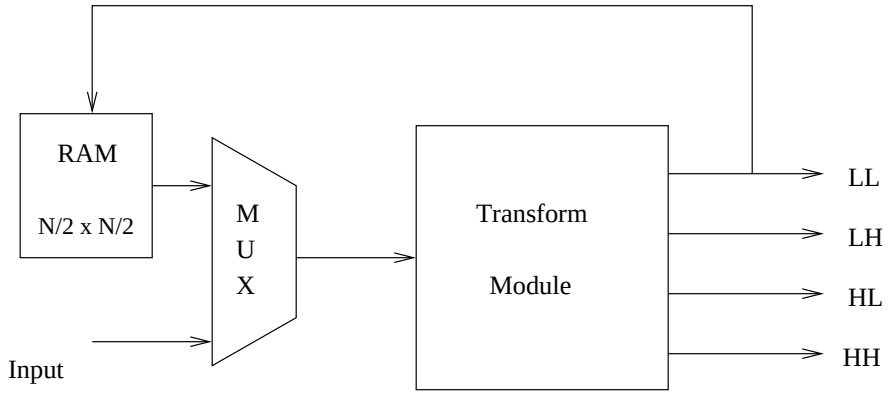

Figure 3.1: Block diagram of Level by Level architecture

The transform module had a tree structured design and was comprised of first the horizontal filtering stage and then the vertical filtering stage as shown in figure 1.4. Generally, the level by level architecture is inefficient as the vertical filtering stage remains idle during row filtering. The design proposed in this paper tried to avoid this by making use of two types of architectures for the decimation filter in each stage.

The first architecture was based on the polyphase decomposition of the filter coefficients into even parts and odd parts. In this method, the input data was fed to the odd part and multiplied with the odd ordered coefficients in the even cycles. While in the odd clock cycles, the input data was fed to the even part and multiplied with the even ordered coefficients. The output data was the sum of the even and odd parts. The internal clock rate was half the input clock rate which reduced the computational processing time by half.

The second architecture was based on a coefficient folding technique, where two coefficients shared a multiplier, an adder and a shift register. The multiplexers or switches were used to control the data coming in. The execution of the data computations were ordered as required in the convolutional filtering approach and the correct result was obtained. The area was also reduced by half due to the sharing of resources.

It was shown experimentally that the throughput was improved resulting in 100% hardware utilization when the polyphase technique was used in stage 1 and coefficient folding technique was used in stage 2 [19].

Kishore Andra and et al., [3] also proposed a level based architecture for the 2D-DWT using the lifting based scheme. The proposed architecture computed the multilevel DWT, one level at a time. It consisted of two row processors and two column processors. Each pair of processors used a register file in between them for data communication and the outputs generated by these processors were stored in memory modules. The block diagram of the proposed architecture is shown in the figure 3.2.

In order to use the same design, the paper classified wavelet filters depending on the number of lifting steps into two types - 2M filters having two lifting steps and 4M filters having four lifting steps. Precision analysis carried out on the filters showed that the maximum signal to noise ratio can be obtained for 256 gray scale images if the 8 bit input values were scaled to 13 bits, filter coefficients were represented using 10 bits and the data path was fixed at 16 bits.

In the case of the 2M filters, the row processors read in the data from the external memory, performed the DWT along the rows and wrote the data into the column memory module. One of the column processors read the data from its memory module, performed the column DWT on alternate rows and wrote back the result to the column memory module. The second column processor read the data processed by the first column processor, performed the DWT along the rows on which the first column processor did not work and wrote the results to the row memory module and the external memory.

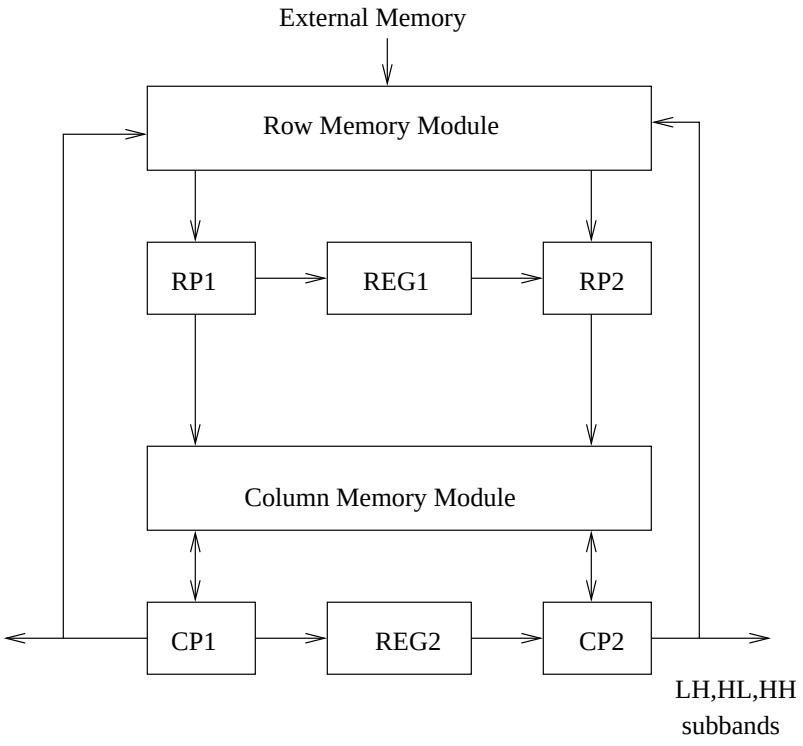

Figure 3.2: Block diagram of Lifting based Level by Level Architecture

In the case of the 4M filters, each DWT along the rows and the columns consisted of four lifting steps. The rowwise 1D-DWT was done in the first pass with each of the row and column processors performing one lifting step. The results of the first pass were written in the row memory module. In the next pass, the transform was computed along the columns and the LL subband was stored in the row memory module and other subbands in the external memory.

3.3 Line-based Architecture

Line based architectures usually consist of separate processors to do the row and the column processing. The column processor is started when the minimum number of processed rows required for the column processing is available. This helps to avoid the use of an external memory to store the results in a single level between the row and the column processing. A local memory with space enough to hold the appropriate number of rows is used instead of the external memory.

Chao-Tsung Huang et al., proposed a line based arhitecture using the lifting method in their paper [11]. The proposed architecture consisted of a row module, column module, an intermediate line buffer and a temporal data buffer. The processor modules generated the 1D-DWT of the input data using the lifting scheme. Each of the processor modules had a systolic architecture derived from the dependence graph corresponding to the lifting scheme. The intermediate buffer had six separate two port memories with half the signal width. Three of the memory ports were used for storing row filtered low pass signals and three for storing row filtered high pass signals. These intermediate memories acted as dynamic rotation buffers. In each clock cycle, the column filter module read from two of the intermediate memories of low pass or high pass signal and the row filter module wrote to one pair of intermediate memories. The temporal data buffer was part of the 1D-DWT architecture in the processor modules and was used as temporal registers in the 1D-DWT computations.

The recursive pyramid algorithm (RPA) proposed by M. Vishwanath [18] can be used in a line based architecture to increase the throughput. The use of the RPA might result in the increase of the local memories used inside the design. The block diagram depicting a generalized line based architecture is given in figure 3.3.

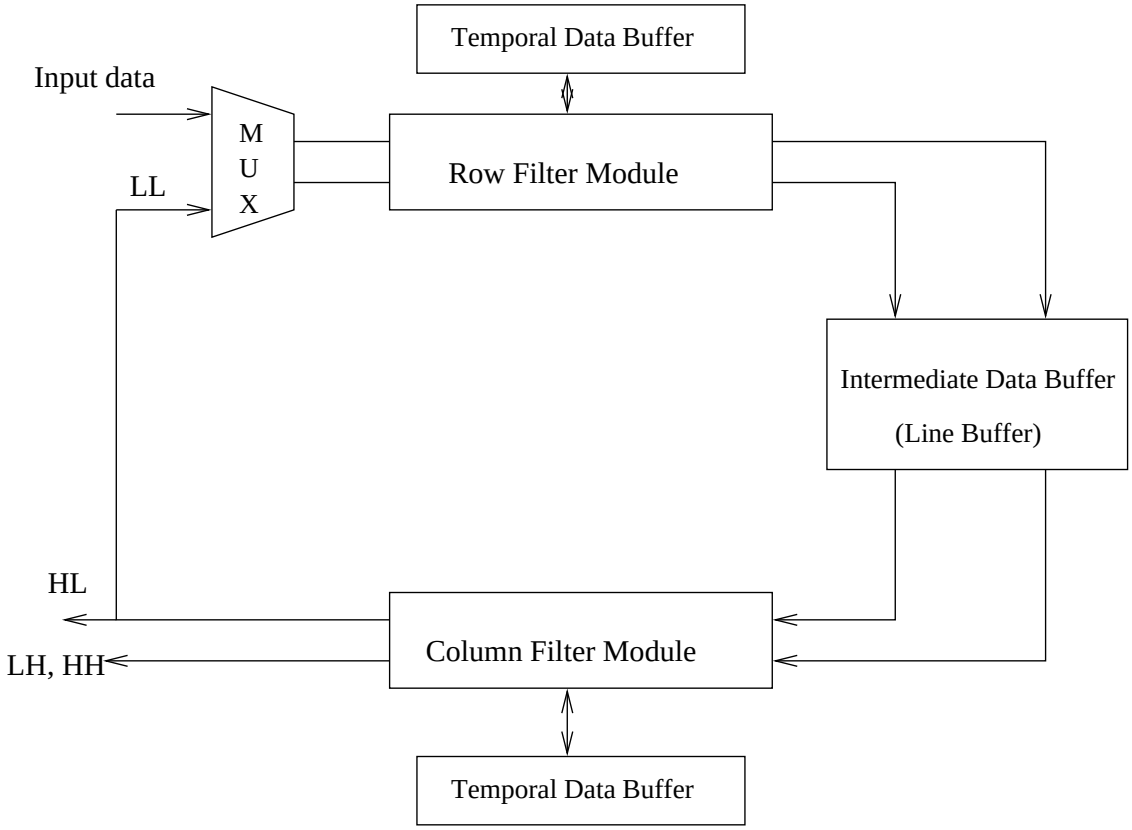

Figure 3.3: Block diagram of Line based architecture

S. Barua et al., proposed in their paper [4] a 2D-DWT architecture which was a hybrid of level by level and line based architectures. The architecture used a local memory between row and column processing in a single level and used the external memory between the levels. The external memory was thus read from and written to in a row wise order.

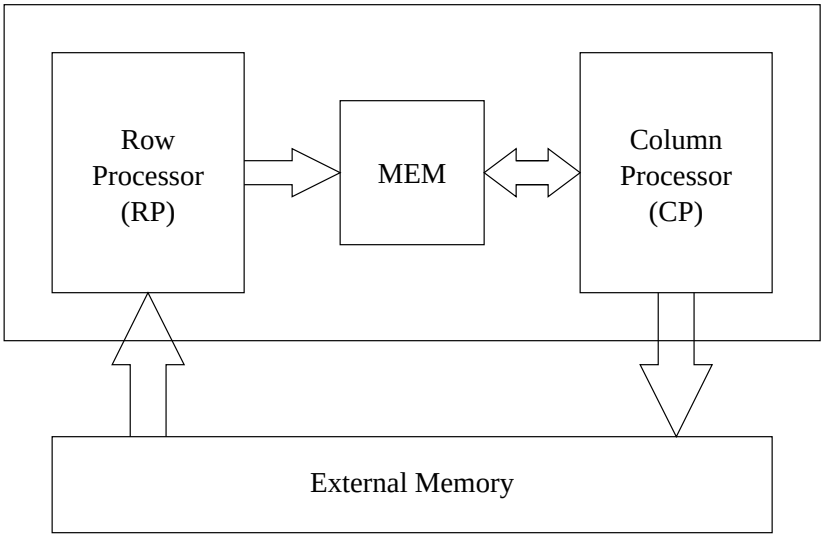

Figure 3.4: Block diagram of Hybrid system architecture

The block diagram of the proposed architecture is shown in figure 3.4. It consisted of a row processor, a column processor and a local memory between the processors. The image in the external memory was read row wise and processed by the row processor. The resulting aproximation and detail coefficients were written into the local memory. The row processor used a systolic architecture to implement the lifting scheme for the computation of the 1D-DWT whereas the column processor used a parallel architecture.

Barua and et al showed the 2D-DWT architecture using biorthogonal (9,7) wavelet filter as an example. Since the (9,7) filter had four lifting steps and two normalization steps, the row processor had six computing modules with each module performing one of the steps. Each module used a systolic architecture to do the computation and the cascaded modules were used to implement the whole lifting structure. In each clock cycle, the row processor read in a pair of data in even and odd positions of a row and produced a pair of outputs. The approximation and detail coefficients from the row processor were stored in the local memory. Since the (9,7) filter was used, at least 9 rows should normally be buffered before

column processing can start. The filter moved one column to the right in each clock cycle. The proposed architecture exploited the lifting structure and the column processor started functioning as soon as three rows of horizontally filtered data was available in the local memory. This was because only three values in a column were needed to start the first lifting step. The remaining pixels were used in the next lifting steps. This also helped to reduce the size of the local memory and a buffer space for only seven rows was needed.

In the time it took the row processor to complete filtering one row of data, the column processor finished processing half the columns in a row. Hence when the column processor finished processing all the columns in a row and skipped two rows to start the next processing, the row processor also completed processing two rows. Thus the row and column processors remained synchronized.

3.4 Block-based Architecture

The block based architectures usually read in the image block by block. They require additional computations at the boundaries to take care of the data dependencies with other blocks. The block of the image read in is usually stored in an embedded memory of the processor and the coefficients are generated at the highest level of decomposition with each block.

Antonio Ortega and Wenqing Jiang [12] proposed a block based parallel architecture to compute the DWT using the lifting scheme. The architecture used a finite state machine model of the DWT derived from the lifting scheme. Each sample in the block was updated to the final state using the samples from the neighbourhood in the DWT using lifting. The samples from neighboring blocks were needed to completely update the samples at the boundaries. This block based architecture used a novel technique based on the overlap-add scheme for the DFT computations to get the correct results at the boundaries. The samples at the boundaries were also updated and changed into intermediate states, these partially updated samples were stored as the state information and processed in the final stage.

In this architecture, there were two stages of operations - Split and Merge. The split stage and merge stage of proposed parallel architecture is illustrated in figures 3.5 and 3.6. In the split stage, the input data was uniformly segmented into non overlapping blocks

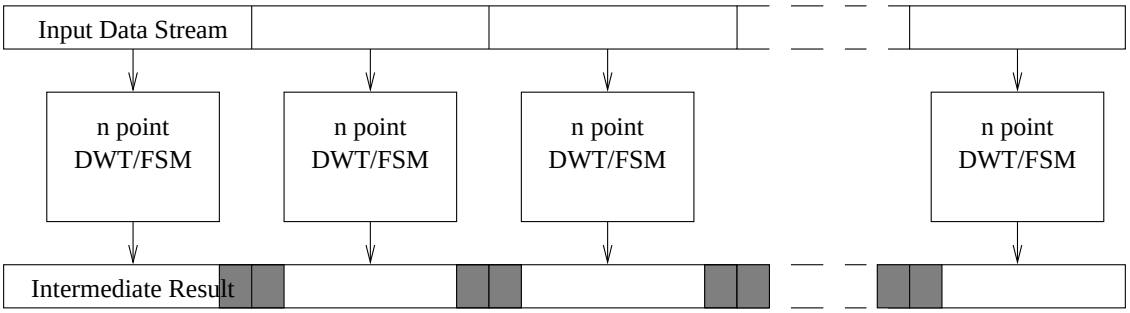

Figure 3.5: Split stage in the block based architecture

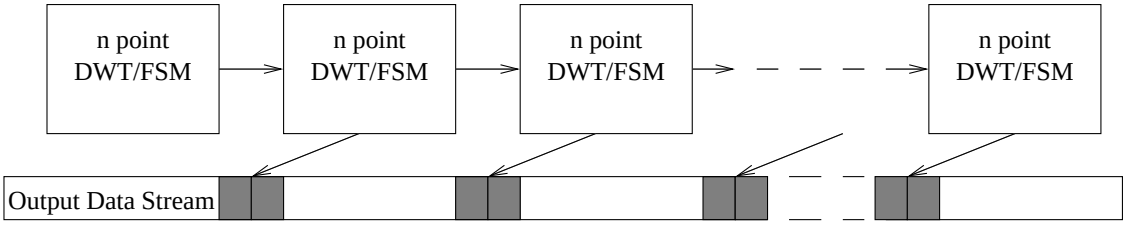

Figure 3.6: Merge stage in the block based architecture

and were allocated to p processors. Each processor computed the DWT of the input data up to the required decomposition level. At the end of the computations, each processor gave as output the completely updated transform coefficients and the partially updated coefficients called the state information. In the merge stage, the state information was communicated between the neighboring processors. At the end of the merge stage, each processor used the information it had along with the information from it's neighbours to give the completely updated coefficients.

Chapter 4

Algorithm Design of Two Dimensional Wavelet Transform

4.1 Introduction

The objective of this thesis was to develop a hardware model for the decomposition and reconstruction of images for telemedicine application using the lifting scheme. The design of a hardware model involves the following phases

1. Requirement Specifications: In this phase, the functional description and the performance specification requirements of the system to be developed is captured.

2. Algorithm Design: This phase involves development of the algorithm that will do the functions required by the system. The capture of the algorithm using a high level language and simulation of the algorithm to verify the correctness of the design is also done in this step.

3. Hardware Behavioral Model: The behavioural model of the system in hardware is usually designed in a high level language (eg: Matlab in this thesis) to help the register transfer level coding easier. The model is then recaptured using a register transfer language (RTL) (eg: this thesis uses Verilog). The simulation outputs from

the RTL design is compared with the results from the algorithm design to verify the correctness of the design[2].

This chapter presents the initial two phases in hardware modeling, the requirement specifications and the algorithm design. The algorithm for the forward and inverse 2D-Discrete Wavelet Transform for use with region of interest coding of medical images was verified by the implementation in Matlab.

4.2 Requirement Specifications

In chapter 1, it was mentioned that the images used in telemedicine should adhere to stringent quality requirements to prevent any errors in diagnosis. The DWT implementation with floating point hardware will yield better quality. However, the usage of floating point hardware models have several disadvantages such as the need for larger amount of hardware, the higher cost in the hardware manufacture and the lower throughput. These drawbacks motivate the use of an integer model where the hardware involved is less complicated. The integer model has the disadvantage that the precision is lost due to quantization. This leads to higher values of round off error in the reconstruction of the images. The integer to integer wavelet transform (IWT) presented in section 2.3 can be used to overcome this drawback. In the IWT, the lifting structure is modified to reduce round off error and yield perfect reconstruction for integer inputs such as image data. Even though each lifting step involves floating point computations, quantization is done in each step and the round off error in the final result is reduced.

The objective was thus to develop a hardware model which does the wavelet decomposition and reconstruction of images with minimum round off error using the integer to integer wavelet transform. There were two major bottle-necks in the design of the hardware model. One of the difficulties was that the lifting steps are anticausal, it involves the use of not only the current and past inputs but also the future inputs. The initial section of this chapter explains the two designs developed as part of this thesis that overcame the difficulty of the anticausal lifting structure. The second bottle-neck was the floating point computations in each lifting step. The floating point coefficients used in each lifting step were scaled to integers to develop the integer model for reducing hardware cost. The coefficient scaling had to avoid the results of computation from overflowing the fixed data path

used inside the system. The latter part of this chapter shows the coefficient scaling method used in this thesis to avoid the overflow of the datapath.

4.3 Implementation of the Lifting Structure

The difference equations 4.3, 4.4, 4.5 and 4.6 show the lifting steps to implement the filtering operation using the (9,7) biorthogonal wavelet filter. However, equations 4.3 and 4.5 show that the current and future values of the input $(s_0[n], s_1[n])$ are used in the calculation of the high pass component of the wavelet transform using the (9,7) filter. Hence to solve this problem two approaches were used. The first approach was to develop a pipelined state space model for the lifting structure. The second approach used the data dependency graph for the lifting wavelet filter structure to develop a more efficient model.

4.4 State Space Model for the Lifting Structure

In a state space model, the contents of the delays/registers are equivalent to current state information. The output at a time instant can depend only on the inputs at that instant and/or the present state information in the model. The difference equations for the lifting steps for the (9,7) filter were delayed by appropriate values in order to develop a state space model to be implemented in hardware. The original lifting difference equations and delayed difference equations are shown below.

The original equations are

$$s_0[n] = x[2n] \tag{4.1}$$

$$d_0[n] = x[2n+1] \tag{4.2}$$

$$d_1[n] = d_0[n] - \alpha_0 * (s_0[n+1] + s_0[n]) \tag{4.3}$$

$$s_1[n] = s_0[n] - \alpha_1 * (d_1[n] + d_1[n-1]) \tag{4.4}$$

$$d_2[n] = d_1[n] + \alpha_2 * (s_1[n+1] + s_1[n]) \tag{4.5}$$

$$s_2[n] = s_1[n] + \alpha_3 * (d_2[n] + d_2[n-1]) \tag{4.6}$$

where $x[n]$ is the input signal, $s_2[n]$ and $d_2[n]$ are the low pass and high pass components respectively.

The delayed equations are

$$\begin{aligned} rd_1[n] &= d_1[n-1] \\ &= d_0[n-1] - \alpha_0 * (s_0[n] + s_0[n-1]) && (4.7) \\ rs_1[n] &= s_1[n-1] \\ &= s_0[n-1] - \alpha_1 * (d_1[n-1] + d_1[n-2]) \\ &= s_0[n-1] - \alpha_1 * (rd_1[n] + rd_1[n-1]) && (4.8) \\ rd_2[n] &= d_2[n-2] \\ &= d_1[n-2] + \alpha_2 * (s_1[n-1] + s_1[n-2]) \\ &= rd_1[n-1] + \alpha_2 * (rs_1[n] + rs_1[n-1]) && (4.9) \\ rs_2[n] &= s_2[n-2] \\ &= s_1[n-2] + \alpha_3 * (d_2[n-2] + d_2[n-3]) \\ &= rs_1[n-1] + \alpha_3 * (rd_2[n] + rd_2[n-1]) && (4.10) \end{aligned}$$

In real time applications we use bounded input sequences, the computations at the boundaries of such sequences using the difference equations have to satisfy certain conditions. Assume that the even sequence $s_0[n]$ and the odd sequence $d_0[n]$ are obtained from a one dimensional sequence having $2l$ elements. Then $s_0[n]$ and $d_0[n]$ are l element sequences, having elements from time index $n = 1$ to time index $n = l$. This implies that $s_1[n]$, $d_1[n]$, $s_2[n]$ and $d_2[n]$ are all l element sequences. From the delayed equations obtained for the state space model, we find that at $n = 1$, $rd_1[1] = d_1[0]$ and $rs_1[1] = s_1[0]$. This exceeds the left bound of bounded sequences $s_1[n]$ and $d_1[n]$, therefore the values of rs_1 and rd_1 at $n = 1$ should be set to zero and the computation for the wavelet coefficients should only start from $n = 2$.

There are l values of s_1 and d_1 to be computed. However, from equations 4.8 and 4.9, we find that $rd_1[l+1] = d_1[l]$ and $rs_1[l+1] = s_1[l]$. Therefore $l+1$ values of $rd_1[n]$ and $rs_1[n]$ have to be computed to obtain the l values of $s_1[n]$ and $d_1[n]$. At $n = l + 1$,

$$rd_1[l+1] = d_0[l] - \alpha_0 * (s_0[l] + s_0[l+1]).$$

Thus the calculation of $rd_1[l+1]$ uses $s_0[l+1]$ which exceeds the right bound of bounded sequence $s_0[n]$. Hence we extend the sequences s_0 and d_0 by adding one zero at the end.

Similar special cases at the boundaries have to be taken care of in equations 4.10 and 4.10 for the computation of $rd_2[n]$ and $rs_2[n]$.

4.5 Pipelined Architecture using a State Space Model for the Biorthogonal (9,7) Filter

The block diagrams in figures 4.1 and 4.2 show the same state space model for the forward wavelet transform, using the (9,7) filter as given by the above steps, drawn in two different ways.

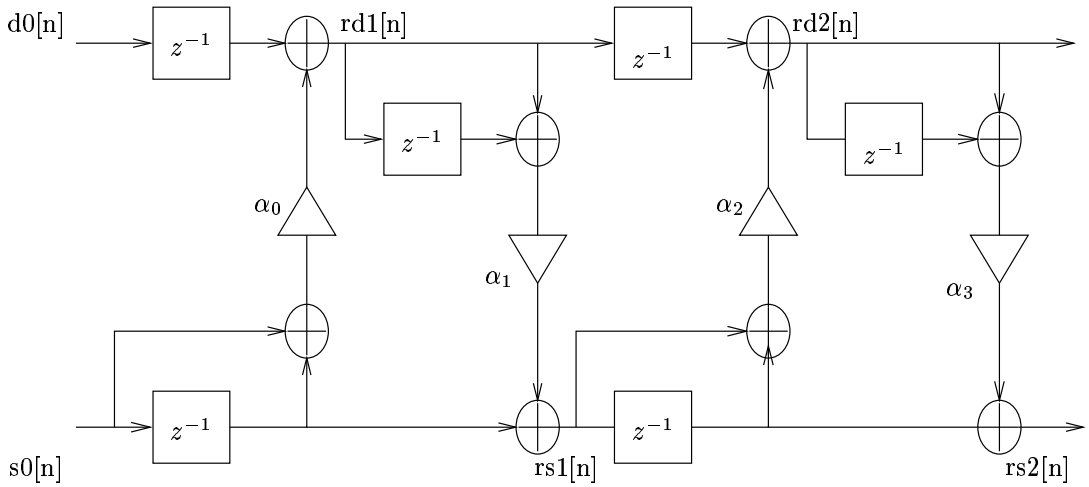

Figure 4.1: State Space Model 1 for Forward Wavelet Transform

It would be easier to develop a pipelined architecture for a given model, if we could find a repeating computational cell. Then we could apply retiming techniques to construct a pipelined model of the computational cell which could be repeated to construct the hardware model for the filter. The similarity in predict steps (equations 4.8 and 4.9) and the similarity in the update steps (equations 4.10 and 4.10) showed that repeating cell structures exist in the state space model. The model given in figure 4.1 shows the two types of repeating cell structures in the predict and update steps.

The state space model for the (9,7) filter in figure 4.1 uses two types of cell structures - a predict cell and an update cell. The block diagrams in 4.3 and 4.4 shows the

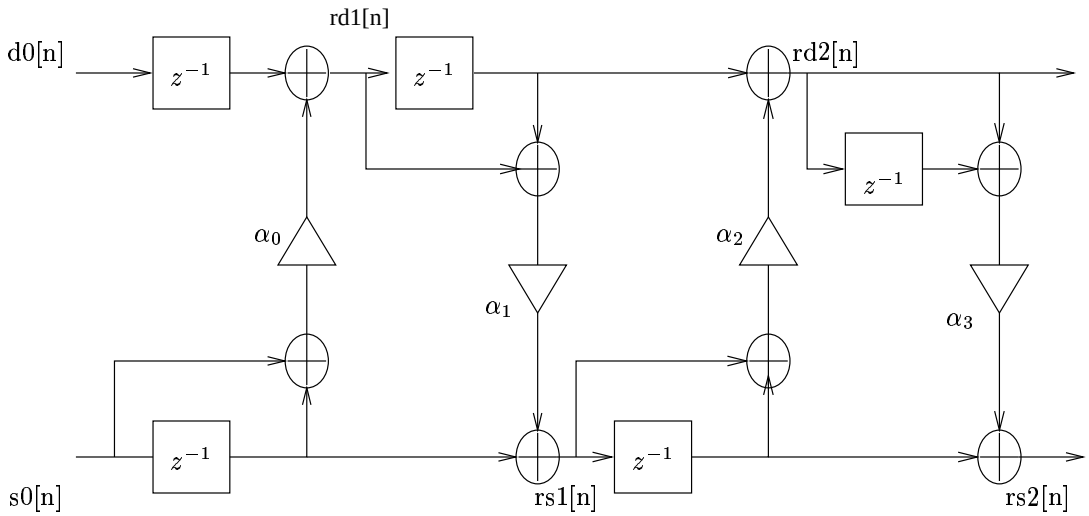

Figure 4.2: State Space Model 2 for Forward Wavelet Transform

predict and update cell structures.

The pipelined architecture for the cell structures of (9,7) filter was developed in this thesis using cutset re-timing to avoid global connections from input to output of the cell. The block diagrams in 4.5 and 4.6 shows the pipelined structures obtained after cut set retiming across the delays in the state space model cell structures.

4.6 Matlab Integer Model for 1D Wavelet Transform

It was explained in section 2.3, how to obtain an integer to integer wavelet transform using the lifting structure. This transform is of particular use in image processing applications where the input image data are integers. In each lifting substep, the result obtained after passing through the update or predict filter coefficient can be rounded as integers before adding or subtracting.

Even though the results of filtering can be rounded as integers, the (9,7) filter uses irrational floating point filter coefficients. The use of floating point operations makes the computations complex and slower in hardware. It is better to use fixed point data representation rather than floating point representation of the coefficients to achieve computational efficiency in the hardware. Fixed point data representation and integer rounding off at the

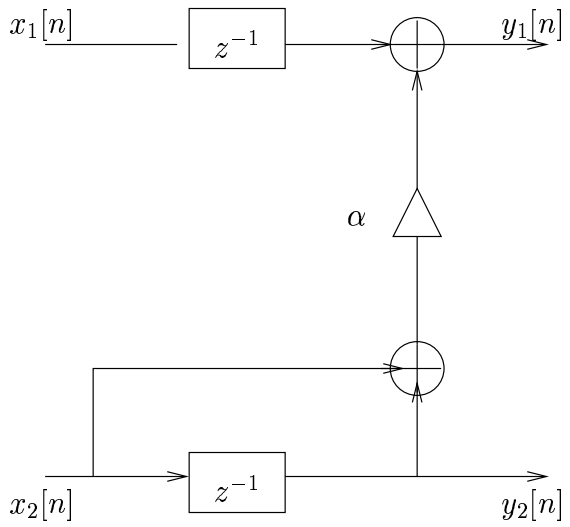

Figure 4.3: Block diagram of Predict Cell

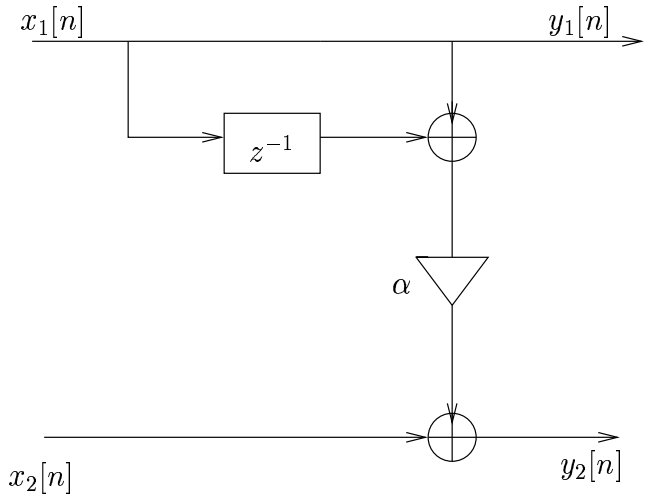

Figure 4.4: Block diagram of Update Cell

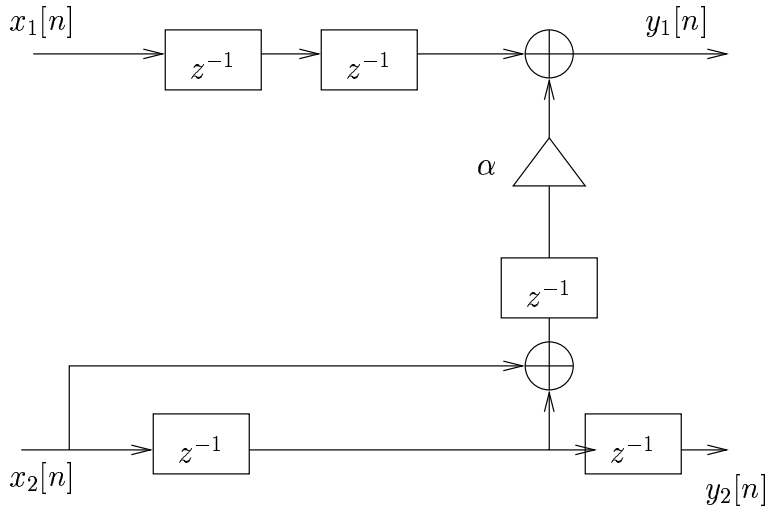

Figure 4.5: Block diagram of Predict Cell after cut set retiming

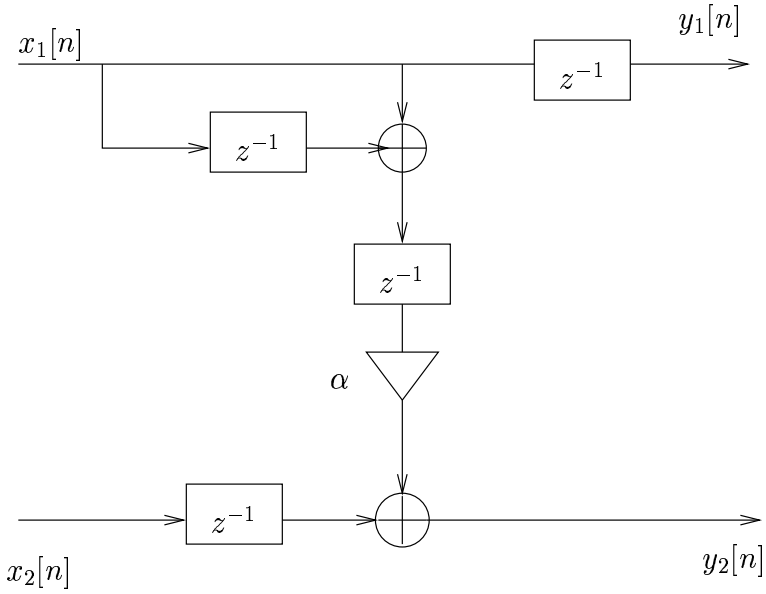

Figure 4.6: Block diagram of Update Cell after cut set retiming

end of each lifting step was used to design the Matlab integer model for the (9,7) filter in this thesis. These design strategies made the computation faster when the model was implemented in hardware.

In the design of the model, the following assumptions were made.

1. The inputs and outputs to the computational cell used 16 bit ports.

2. Each computational cell used a 16X16 multiplier and to maintain fixed data path of 16 bits the output of the multiplier was left shifted.

The filter coefficients were scaled to 16 bits by multiplying each coefficient by 2^{14}. At the end of the multiplication, the product (32 bits) was scaled down to 16 bits by left shifting by 14 bits and losing the 2 MSB's. The input data was assumed to be 8 bits (256 gray scale image) and was represented using sign extended 16 bits. The adders in the computational cell were designed to accept 16 bit values. Since the datapath was fixed at 16 bits through out the model, the design of the hardware was easier.

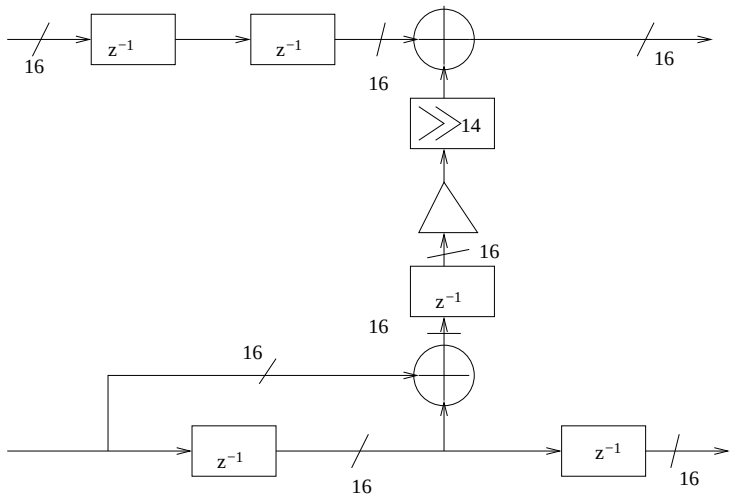

Figure 4.7: Block diagram of predict cell in the integer model of (9,7) filter

It was noted while transforming (256 grayscale) images using the described model that the output dynamic range was not exceeded. In the event, it was exceeded the result would be saturated to 16 bits. Since image data was used, such a saturation would typically be insignificant to the observer after reconstruction.

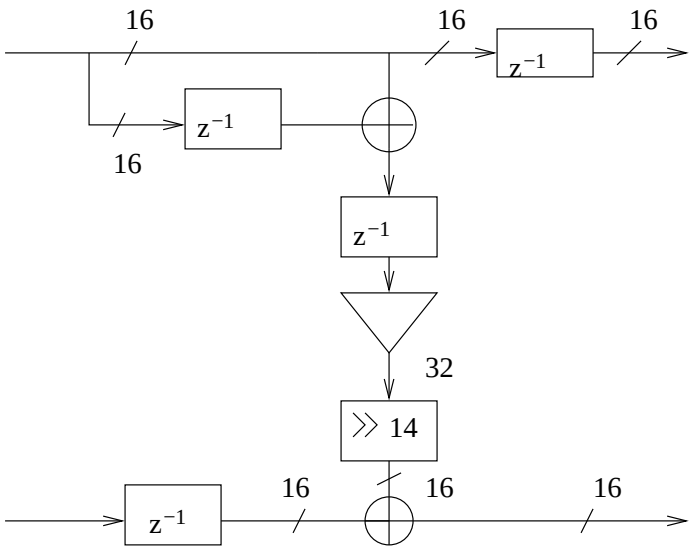

Figure 4.8: Block diagram of update cell in the integer model of (9,7) filter

The reconstruction of the data used the pipelined architecture for the predict cell and the update cell as used in the forward wavelet transform. The block diagram of the state space model for the inverse wavelet transform using the (9,7) filter is as shown in figure 4.9

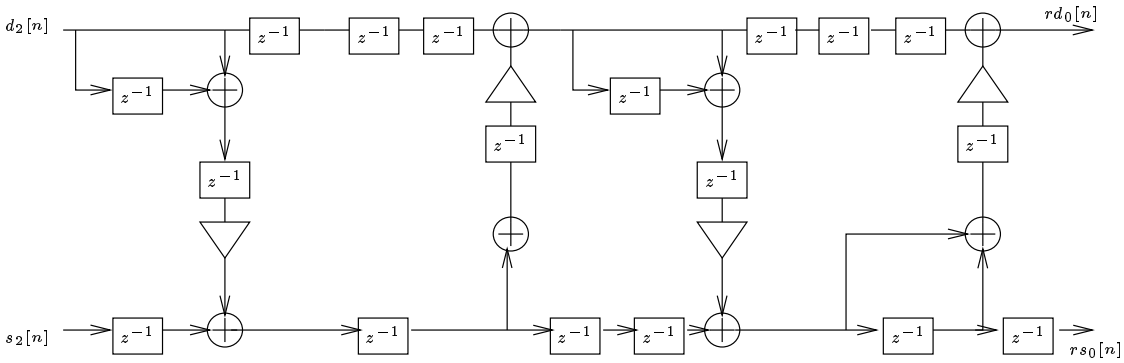

Figure 4.9: State Space Model of Inverse Wavelet Transform

Both the forward and inverse wavelet transforms required twelve additional zeros as inputs to satisfy the boundary conditions mentioned in the section 4.3. The zero inputs

were used to clear the registers before the new line of one dimensional data is read in. If the registers were not cleared, the output wavelet coefficients corresponding to the new line would depend on the previous line of inputs.

4.7 2D Separable Discrete Wavelet Tranform

The two dimensional wavelet transform explained in section 1.1.1 shows that each level of the wavelet decomposition involves two stages - row processing and column processing . If the algorithm does the column wise transform after the row wise transform is completed it is called a level by level algorithm. The level by level architecture thus requires an intermediate memory of the size of the 2D-data being transformed to store the row wise transformed data. The size of the hardware increases with the large intermediate memory and hence, usually the level by level architecture uses the external memory as the intermediate memory. However, using the external memory increases the number of memory accesses which in turn reduces the throughput of the system. The hardware utilization will also be low due to the waiting time for one stage to complete before the next stage can start. Hence, we looked at the use of a 2D-separable transform which started the next stage of processing in a decomposition as soon as the minimum required data for processing was available.

Normally in the 2D-DWT, horizontal filtering is done first followed by vertical filtering. Most of the applications scan the image row wise. We found that if we scan the input image row by row, the memory required if row processing is done first followed by the column processing is about two times the order of the filter used. However, if the column processing is done first followed by the row processing in the 2D-separable transform, the amount of memory required is just equal to the filter order. For example, consider the (9,7) filter used in forward wavelet transform. At least 9 elements are required for processing the data. Hence the row wise transform should process 9 elements in 9 rows to give the 9 row processed elements for the column processing. Since nine approximation and detail coefficients in a column from row wise transforms is needed simultaneously for a column wise transform, 9 rows should be stored for column processing. If the column processing is done first followed by row processing, nine elements from a column from nine rows are needed for column wise transform. For row wise transform, only 9 approximation and detail

coefficients from a single row need to be stored. Thus the system has a lot of memory savings if the column processing is done first followed by the row processing.

Hence the vertical filtering of the data was done followed by horizontal filtering . The horizontal filtering started as soon as sufficient amount of data in a row to start the row processing was available. We found the hardware utilization to be low using the pipelined architecture modelled in section 4.5 even with the use of the 2D separable transform. The pipelined state space model required a minimum of 10 data elements in a column to start the vertical processing. The computation using these 10 elements in a column gave 2 outputs, one low pass coefficient and one high pass coefficient. The vertical processing module gave the next coefficient in the row after reading in 10 data elements from the next column. So the row processing module got the data in the even and odd positions of a row every 20 clock cycles ie., after the vertical module processed two columns of 10 data elements each. The column processing module before processing a whole column jumped to the next column to compute the next coefficient in the row. Hence it could not use the state information from a computation in the next computation which used the next column elements. The row processing module did computation along a row continously unlike the column processing module and thus could use the state information in a computation in the next computation until the end of the row was reached. The throughput was hence at the rate of one output per ten clock cycles which was inefficient. This forced us to look into an alternative design for the lifting structure.

4.8 Model based on the Dependence Graph of the Lifting Structure

The architecture designed as part of this thesis in section 4.5 gave a causal lifting structure and used a fixed point data representation but the throughput was low. A much more efficient model was developed in this thesis to overcome the disadvantages posed by the state space model.

If we look at the data dependency graph for the (9,7) filter forward wavelet transform lifting structure shown in figure 4.10, we can observe a repetitive pattern of data dependency. This repetitive pattern is shown in the figure 4.11. The pipelined architecture

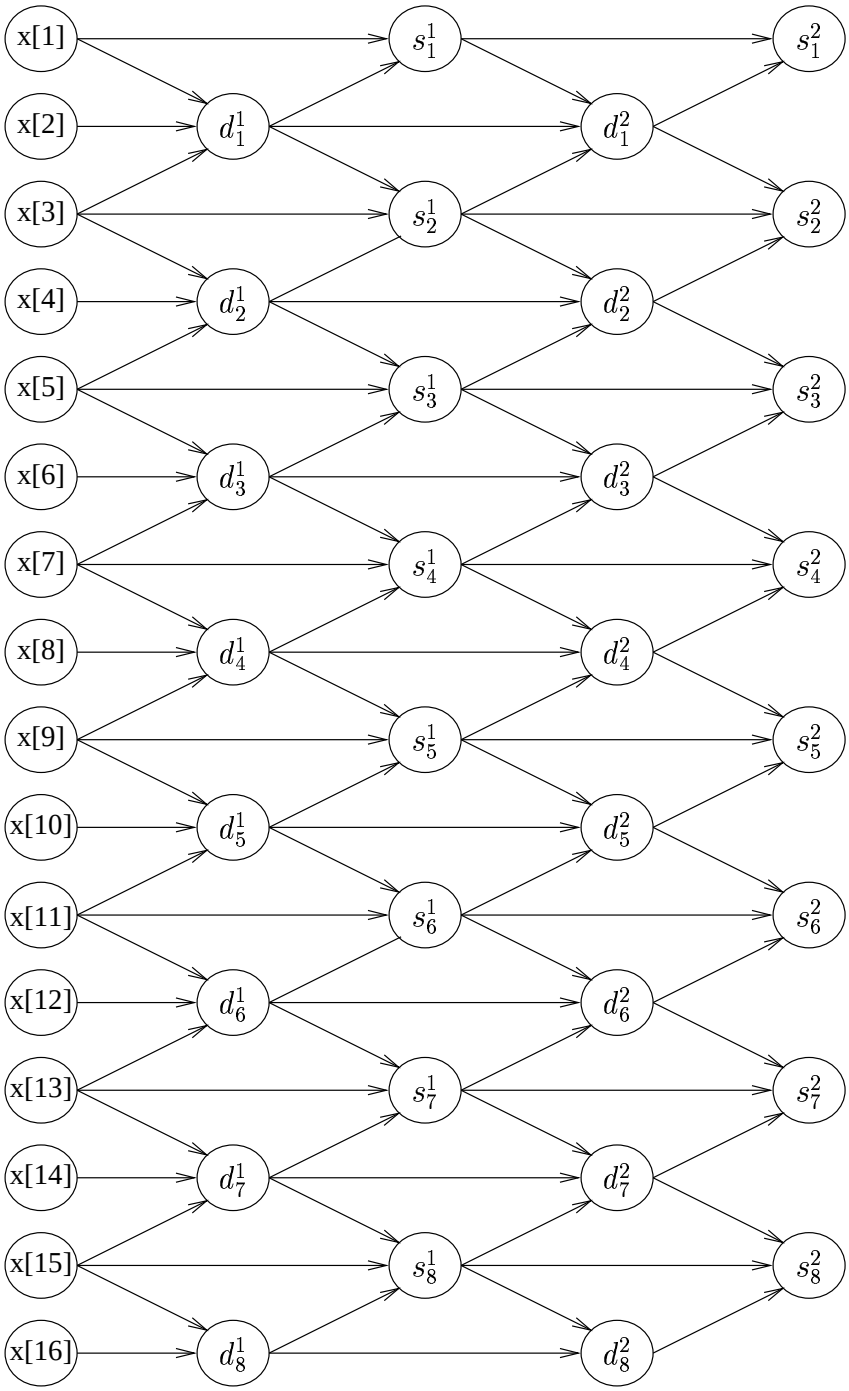

Figure 4.10: Data Dependency graph of (9,7) filter lifting structure

was developed by applying cutset retiming on the data dependency model of the forward wavelet transform given in figure 4.11. The pipelined architecture thus developed is given in figure 4.12.

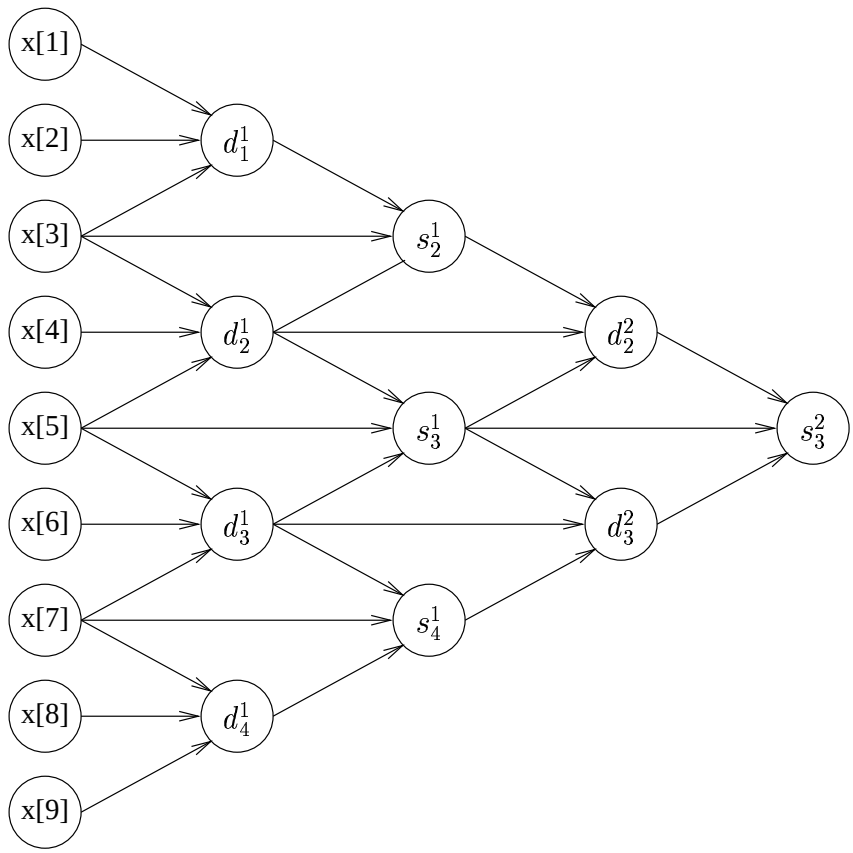

Figure 4.11: Repetitive pattern in (9,7) filter lifting structure

We overcame the inefficiency present in our earlier model where 10 clock cycles were needed to give one approximation and one detail coefficient of a 1D-DWT by using a hardware model with 9 inputs provided at the same time as shown in figure 4.12. Nine separate memory banks were used to store 9 rows of the 2D-data, to efficiently provide the nine inputs needed at the same time. After the initial setup delay, the new model gave outputs every clock cycle. The handling of boundary conditions in the one dimensional DWT was done easily without using the interleaving zeros as needed in the earlier model.

The boundary conditions for the 1D-DWT from the figure 4.10 shows that only

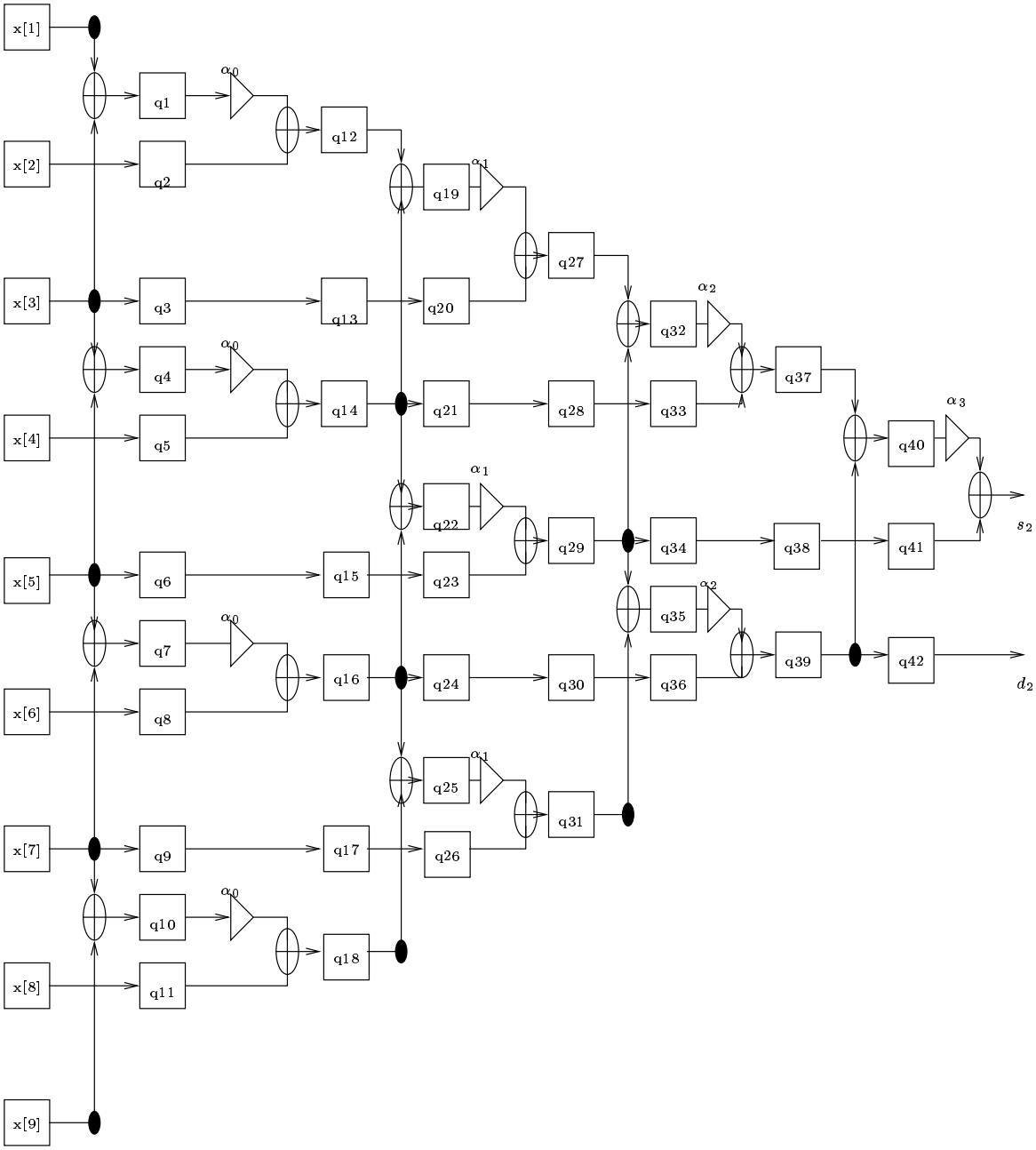

Figure 4.12: Pipelined (9,7) filter lifting architecture

Table 4.1: Order of input data elements in 1D-DWT of numbers 1 to 16

Time Index	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	x[9]
1	0	0	0	0	1	2	3	4	5
2	0	0	1	2	3	4	5	6	7
3	1	2	3	4	5	6	7	8	9
4	3	4	5	6	7	8	9	10	11
5	5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14	15
7	9	10	11	12	13	14	15	16	0
8	11	12	13	14	15	16	0	0	0

5 data elements are needed in order to compute the outputs at the boundaries. So in the hardware model the five data elements at the start of the 1D-DWT was given as the last 5 inputs. Then conditions were set so that registers outside the boundary in this case above $x[5]$ are set to zero for the start of 1D-DWT, thus giving the correct output.

The working of the 1D-DWT model can be explained with the help of an example. Consider that an input data set consisting of numbers from 1 to 16 is given to the model to find the DWT. The table 4.1 shows the order in which inputs are given if the input data is numbers from 1 to 16.

From the table 4.1, we observe a few important points. The input line $x[5]$ and $x[6]$ always has data in each computation. So the boundary registers are registers which do not fall in the same horizontal line as $x[5]$ and $x[6]$. We also observe that the boundary conditions have to be obeyed in time indices 1, 2, 7 and 8. So in order to keep track of the boundary conditions a register is used to store if the boundary case is 1, 2, 7 or 8. And depending on the boundary case, the registers that fall outside the boundary in that particular case are set to zero.

The model for the inverse discrete wavelet transform was developed using the data dependency graph for inverse wavelet transform using (9,7) filter given in figure 4.13. The repetitive pattern present in the data dependency graph is illustrated in figure 4.14. The pipelined architecture for the inverse wavelet transform was developed using cutset retiming as before and the model developed is shown in figure 4.15.

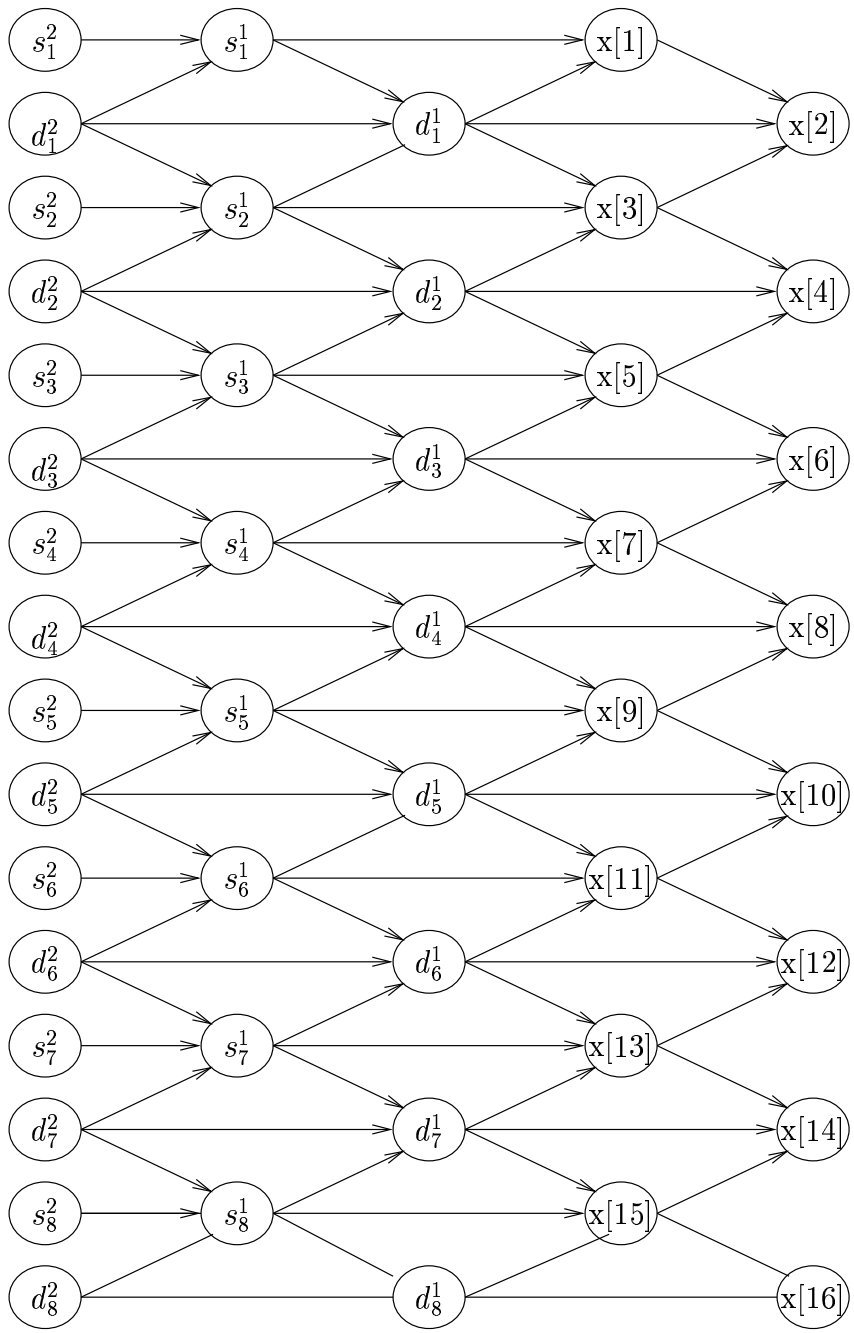

Figure 4.13: Data Dependency graph of (9,7) inverse wavelet filter lifting structure

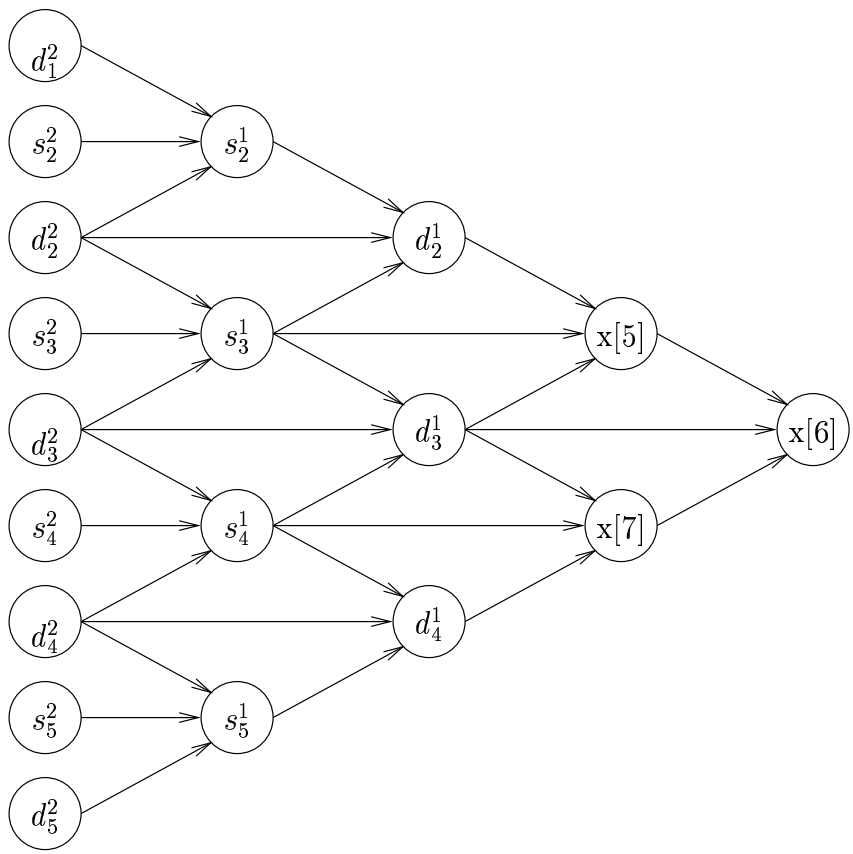

Figure 4.14: Repetitive pattern in (9,7) inverse wavelet filter lifting

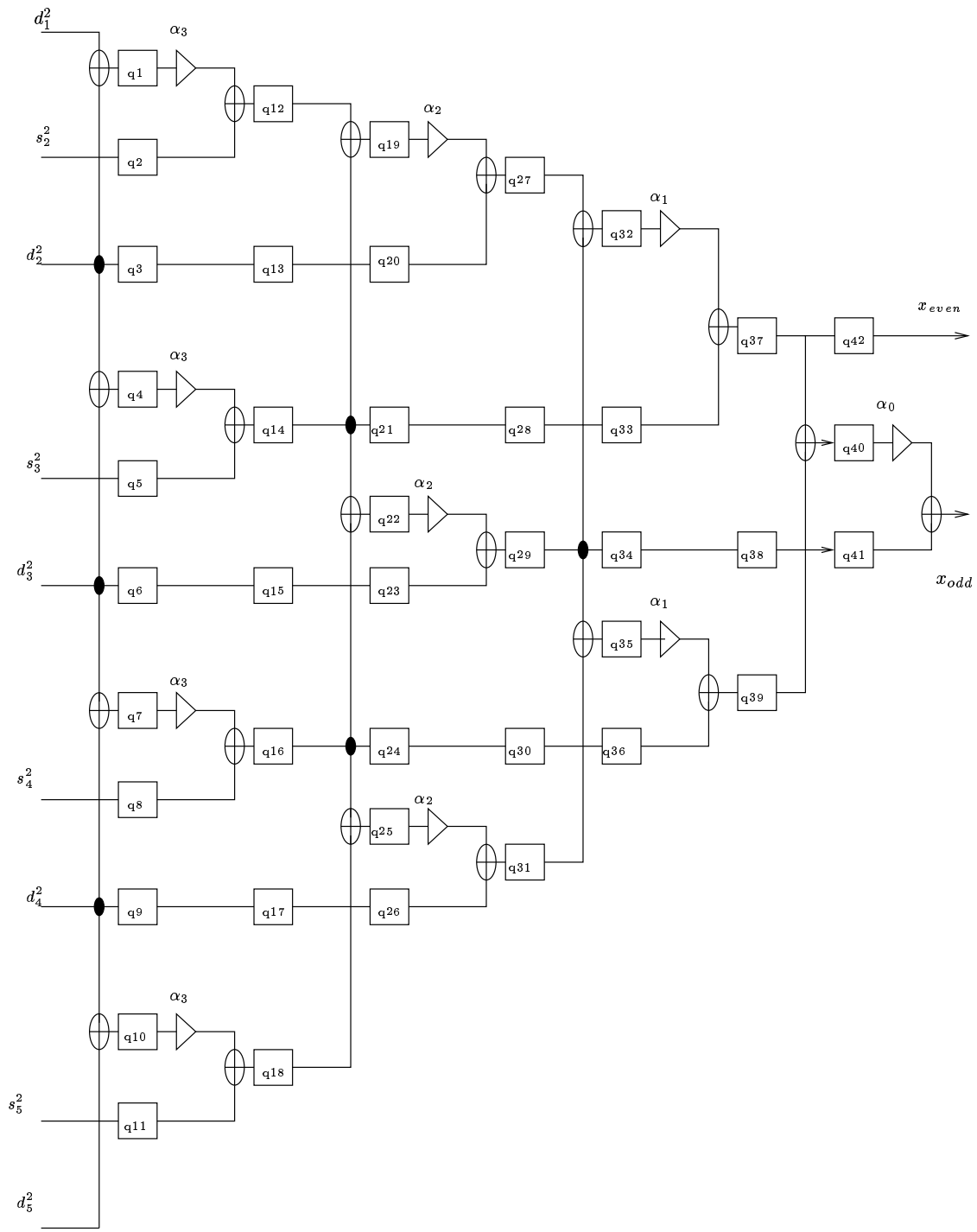

Figure 4.15: Pipelined (9,7) filter lifting architecture for inverse wavelet transform

4.9 Dependence Graph based Integer Model

Section 4.6 explained how an integer model was developed for the state space based design. The same design strategy was reused in the dependence graph based design to get an integer model. All the floating point computations take place at the multipliers in the dependence graph based design. The floating point coefficients were scaled to 16 bits by multiplying each coefficient by 2^{14}. The sign extended 9 bit input data was carried using a 16 bit datapath. All the floating point multipliers in both the forward wavelet transform model and the inverse wavelet transform model were replaced by 16 x 16 multipliers. The 32 bit output of the multiplier was shifted to get 16 bits by losing the 14 LSB's representing the increased precision in the coefficient due to scaling and the 2 MSB's representing the sign extension bits in the output. Thus the model retained a fixed data path size of 16 bits.

4.10 2D Separable Transform using Data Dependency Model

In vertical filtering using this model, 9 data elements in a column of the 2D-data was read at a time. This was done for all the columns from left to right which resulted in one row of approximation coefficients and one row of detail coefficients from the column DWT. After completing one scan from left to right, the column processing skipped two rows, read in two new rows of data and started column processing from left to right. This continued until the vertical filtering of the 2D-data was completed.

The memory banks were initialized with zeros. The boundary conditions for the column DWT from figure 4.10 show that 5 data elements are needed in order to compute the outputs at the boundaries. Since the first 5 data elements in each column were needed to compute the first output for each column, the 5 rows were read in from the external memory and stored in the memory banks before starting the column processing. After storing the 5th row in the memory bank, the column processing was started. During each column processing, the next two row elements were read in from the external memory and stored in the memory banks. Thus after each left to right scan for the column processing, two entire rows of the 2D-data were in the memory bank and were used by the column processing in the next scan.

After obtaining five approximation and five detail coefficients from the column DWT, the row processing was started with these coefficients. The row processing of both

sets of coefficients from the vertical filtering was done at the same time, thus the system gave four outputs one each in LL, LH, HL and HH bands. The model is efficient because after the initial latency due to the pipelining and storing the first 5 rows in the memory banks, we get one output every clock cycle.

Chapter 5

Hardware Behavioural Model for 2D-IWT

5.1 Introduction

In this chapter, we discuss the hardware behavioural model implemented in Matlab and Verilog to capture the algorithm designed in the previous chapter to do the forward and inverse integer wavelet transform of images. The two dimensional separable Integer Wavelet Transform (IWT) was used for the concurrent processing of the columns and rows of the 2D-data. The performance and throughput of the system was improved by the concurrent processing of rows and columns.

5.2 Hardware model for 2D Forward Wavelet Transform

The hardware model for the 2D-Forward Integer Wavelet Transform (FIWT) consisted of three Discrete Wavelet Transform (DWT) modules - one for the column FIWT and two for the row FIWT. The external memory contained the original image of size N x N. There was an intermediate embedded memory bank which stored the incoming data from the external memory and gave the data inputs for the column FIWT.

5.2.1 Column Processing

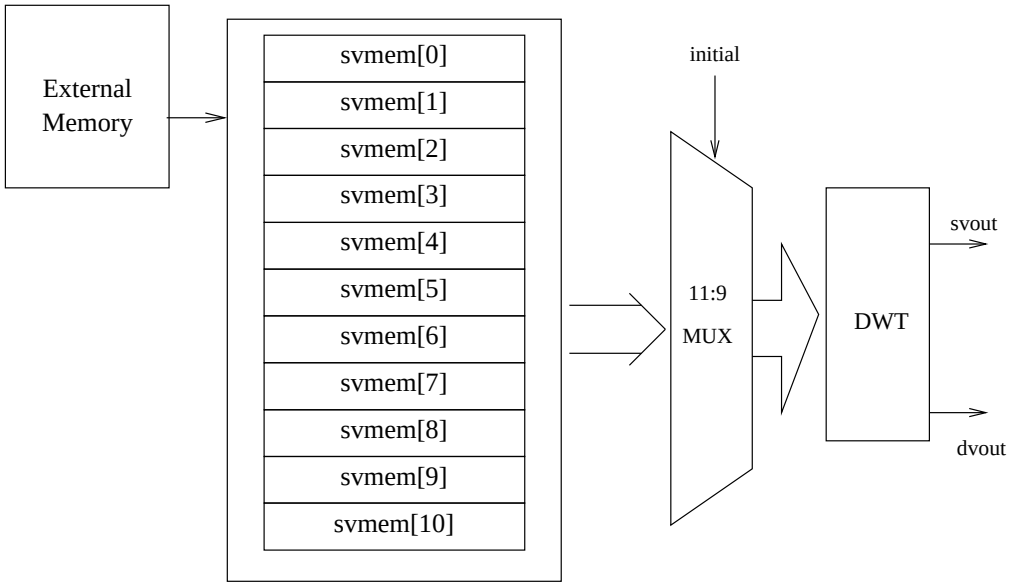

Figure 5.1: Block diagram of Column Processing part of FIWT hardware model

The block diagram in figure 5.1 shows the column processing part of hardware model for the 2D-FIWT. The DWT module used the data dependency graph based pipelined architecture designed in section 4.8. The working of the DWT module is explained in detail in a later section. The architecture required 9 data inputs to be given at the same time. This necessitated the use of an intermediate memory bank with at least 9 memory banks with each memory bank capable of storing one row of the image. The nine memory banks were initially filled with zeros.

Since the boundary computations required a minimum of 5 rows, the first 5 rows of the image were read in from the external memory and stored in five of the memory banks before starting the column processing. The column DWT module started processing after the 5 rows were stored. After completing one scan in which the column DWT data along the entire row was generated, the memory banks read in the next two rows from the external memory. The data elements were read when each column was being processed increasing the efficiency of the system. Two clock cycles were required to read in the two data elements along a column. Thus the DWT module had two clock cycles to do each computation or in

other words the DWT module clock signal had half the rate of the original clock.

If we used only 9 memory banks, then it meant that all the memory banks had to be provided with two data ports - one to read data and one to write data at the same time at different locations. This was because after finishing one column and starting the read for the next column, the next row elements of the previous column had to be read from the external memory and stored in appropriate memory locations of the memory banks. However, the use of 9 two port memory banks is very expensive compared to having two additional memory banks for holding a row of the image. Hence the hardware model was implemented with 11 memory banks with only one data port for both read and write. The use of the additional 2 memory banks eliminated the need to read from and write to the same memory bank at the same time.

The Verilog and Matlab implementations used a large state machine to generate the read enable signals for the appropriate intermediate memory banks. The machine state indicated which memory bank contained the oldest row and which contained the most recent row. The value of the state changed when all the columns in a left to right scan were processed and two new rows were read into the intermediate memory bank. The state machine model used in the implementations is shown in the figure 5.2. The intermediate memory bank thus resembled a FIFO. In the state machine model, the value of the state variable also indicated which memory bank contained the topmost values in the FIFO.

The column DWT module was fed the 9 inputs using a multiplexer. The multiplexer chose the 9 inputs from the 11 inputs it got from the 11 memory banks. The multiplexer used the value of the state variable to find the correct inputs for the DWT module.

There was a latency of 7 clock cycles initially for the column DWT module due to the 7 pipelines inside it. The DWT module generated two outputs for every two inputs read in from the external memory. After the initial latency of 7 clock cycles + (N*5) clock cycles to read in the initial 5 rows, the column processing gave an output for every clock cycle. The approximation and detail coefficients from the column processing were fed into two FIFO register arrays.

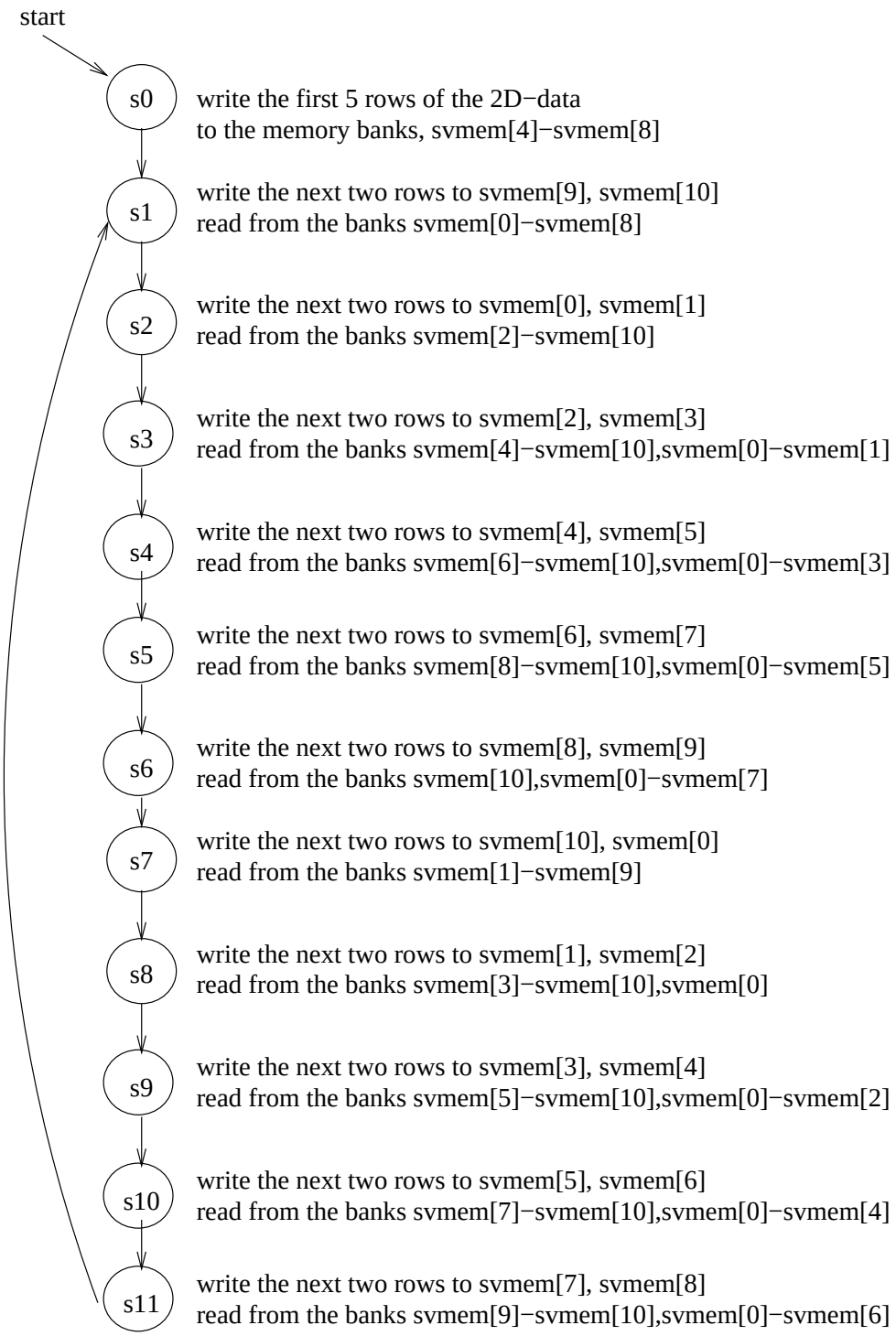

Figure 5.2: FSM used to generate read and write enable signals for the intermediate memory bank in the 2D-FIWT hardware

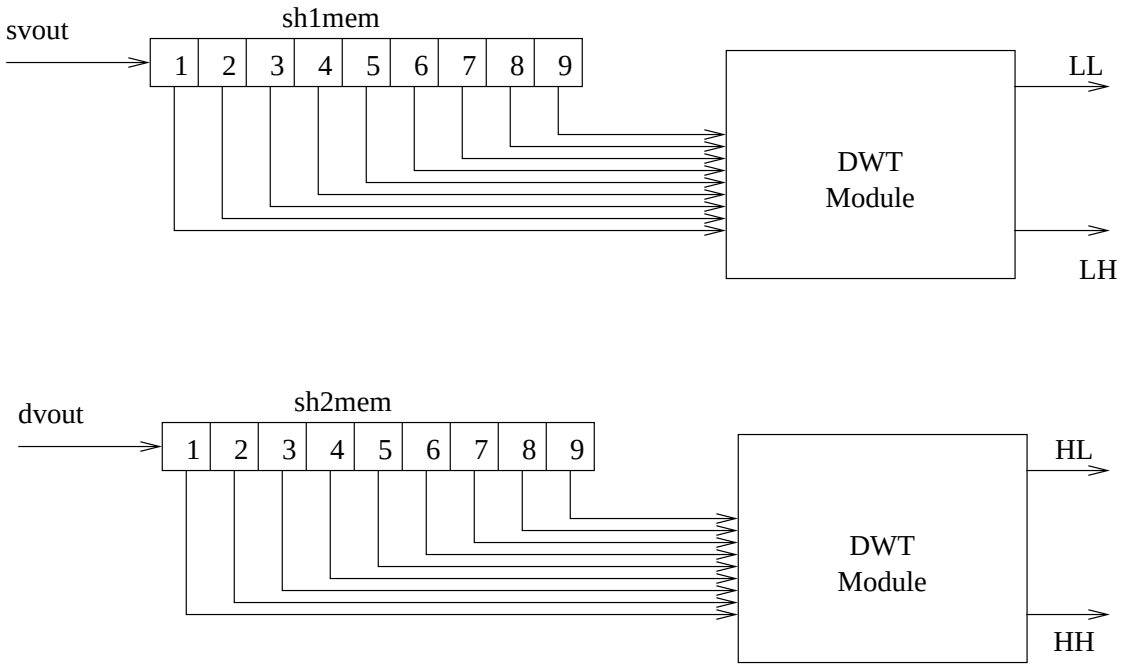

Figure 5.3: Block diagram of Row Processing part of FIWT hardware model

5.2.2 Row Processing

The block diagram in figure 5.3 shows the row processing part of hardware model for the 2D forward integer wavelet transform (FIWT). The row processing DWT modules took in the inputs from the register arrays. The approximation coefficients were given to one DWT module, while the detail coefficients were given to the other module. The row processing could start only after at least 5 elements were available in each register array after column processing. Also the row processing skipped two columns at a time after each computation. Therefore two new coefficients should be available from the column processor before the computation on the row processing resumed. The row DWT module hence worked at half the rate of the column DWT module. However, two row DWT modules were used to process the high pass and low pass coefficients from the column DWT module simultaneously. Thus four outputs were obtained from the two row DWT modules after each computation. So the total throughput of the system remained constant at the rate of one output per clock cycle. The outputs from the two row DWT modules were one coefficient each from the LL, LH, HL and HH band.

5.2.3 DWT module

There were three instances of DWT modules in the implementation of the forward intger wavelet transform. The DWT module used the data dependency graph based pipelined architecture designed in section 4.8. The DWT module gave the low pass and high pass coefficients as outputs. Each DWT module required 20 inputs including the input clock to do the computations. The twenty inputs to the DWT module included the four wavelet lifting filter coefficients, six boundary indicators and the nine inputs used in the computations. The DWT module used to do the vertical filtering received the nine inputs from the external memory and the outputs from these computations were stored in buffers. One buffer stored the low pass coefficient from the vertical filtering while the other stored the high pass coefficient. Each buffer had space to store nine coefficients and these buffer contents were given to the two DWT modules for horizontal filtering.

The data dependency graph hardware model developed as part of this thesis was described in section 4.8. It was explained using a 1D-Integer Wavelet Transform model that in order to compute the correct values at the boundaries, certain conditions had to be

fulfilled. It was also shown that in order to develop the pipelined architecture, seven cut set retimings were used in the dependency graph model. Each cut set retiming introduced a delay or a pipelined stage. All the registers except in the last two pipelined stages had to obey the boundary conditions and hence six registers were needed for each stage to indicate the boundary conditions for that stage. While doing the vertical filtering of the 2D-data, the data in the columns were read one after another. Thus the boundary conditions for the computations were the same for all the columns in a left to right scan and changed only in the next left to right scan. In the row filtering, the computations for the next row started after completing the current row of computations. Therefore the boundary conditions repeated periodically ie., the conditions at the beginning of the row or at the end of the row repeated when the next row computations started.

In order to keep track of the boundary conditions, the column DWT module used six boundary indicator variables and the row DWT modules used a different set of six boundary indicator variables. The six boundary indicators used in both the cases resembled an array of flip-flops with the output value of a flip-flop being passed to the next one at the positive clock edge. Variables were used to keep track of the number of rows being read in the column FIWT and the number of columns used in the row FIWT computations. These variable values were used to set the correct values in the first flip-flop of the array ie., first boundary indicator variable. Figures 5.4, 5.5, 5.6, 5.7, 5.8 5.9, 5.10 and 5.11 show the eight pipelined stages in the DWT module.

5.3 Hardware Model for the 2D Inverse Wavelet Transform

The hardware model for the 2D Inverse Integer Wavelet Transform (IIWT) consisted of four Inverse Discrete Wavelet Transform (IDWT) modules - two for the column IIWT and two for the row IIWT. The external memory contained the transformed image of size N x N. There were two intermediate embedded memory banks which read in data from the external memory and gave the data inputs to the row IDWT modules. Since in the FIWT, the column processing was followed by the row processing, the order was reversed for the inverse integer wavelet transform with the row processing done first followed by the column processing. Since two row modules were used in the forward integer wavelet transform to transform the detail and approximation coefficients from the column processing at

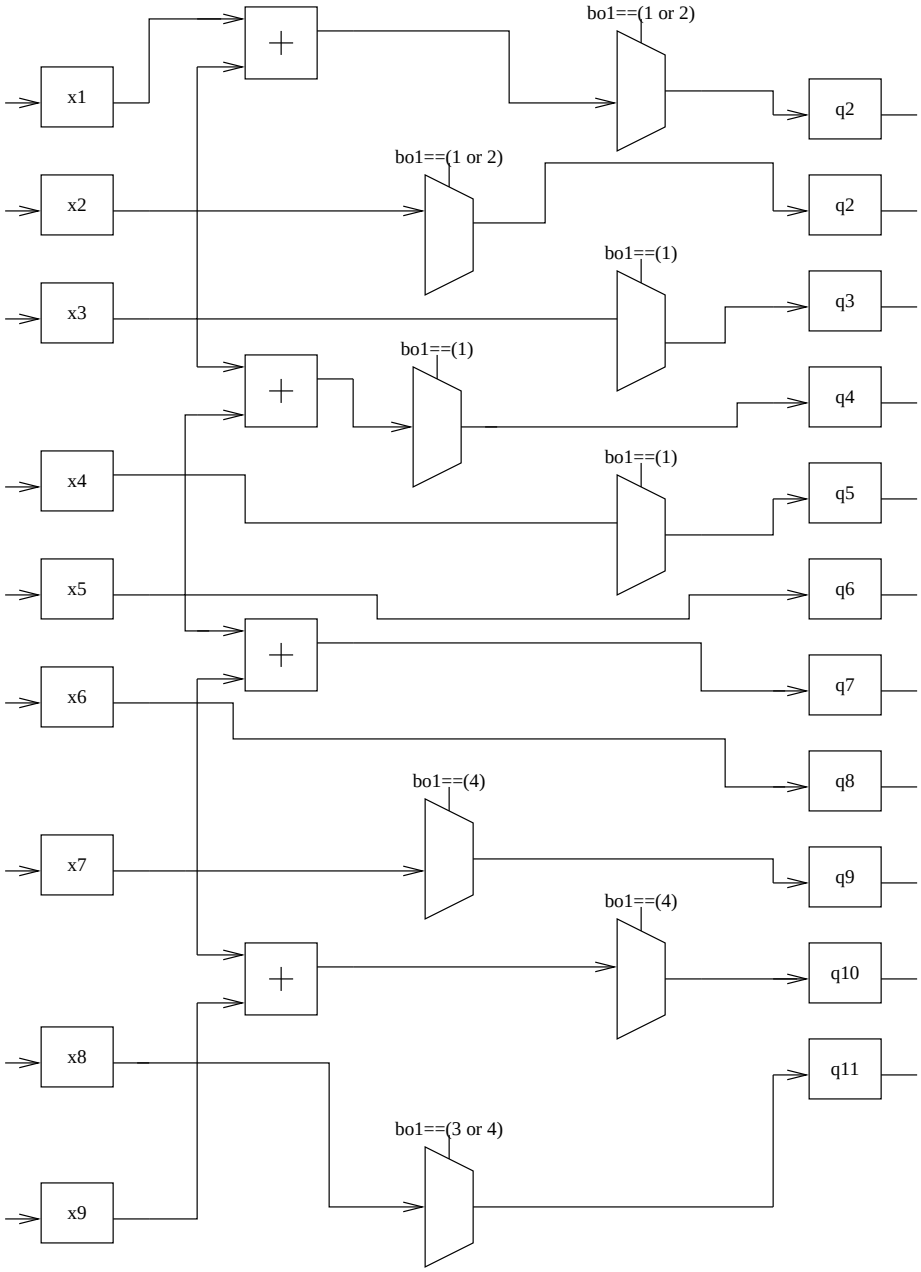

Figure 5.4: First Pipeline stage in the DWT module

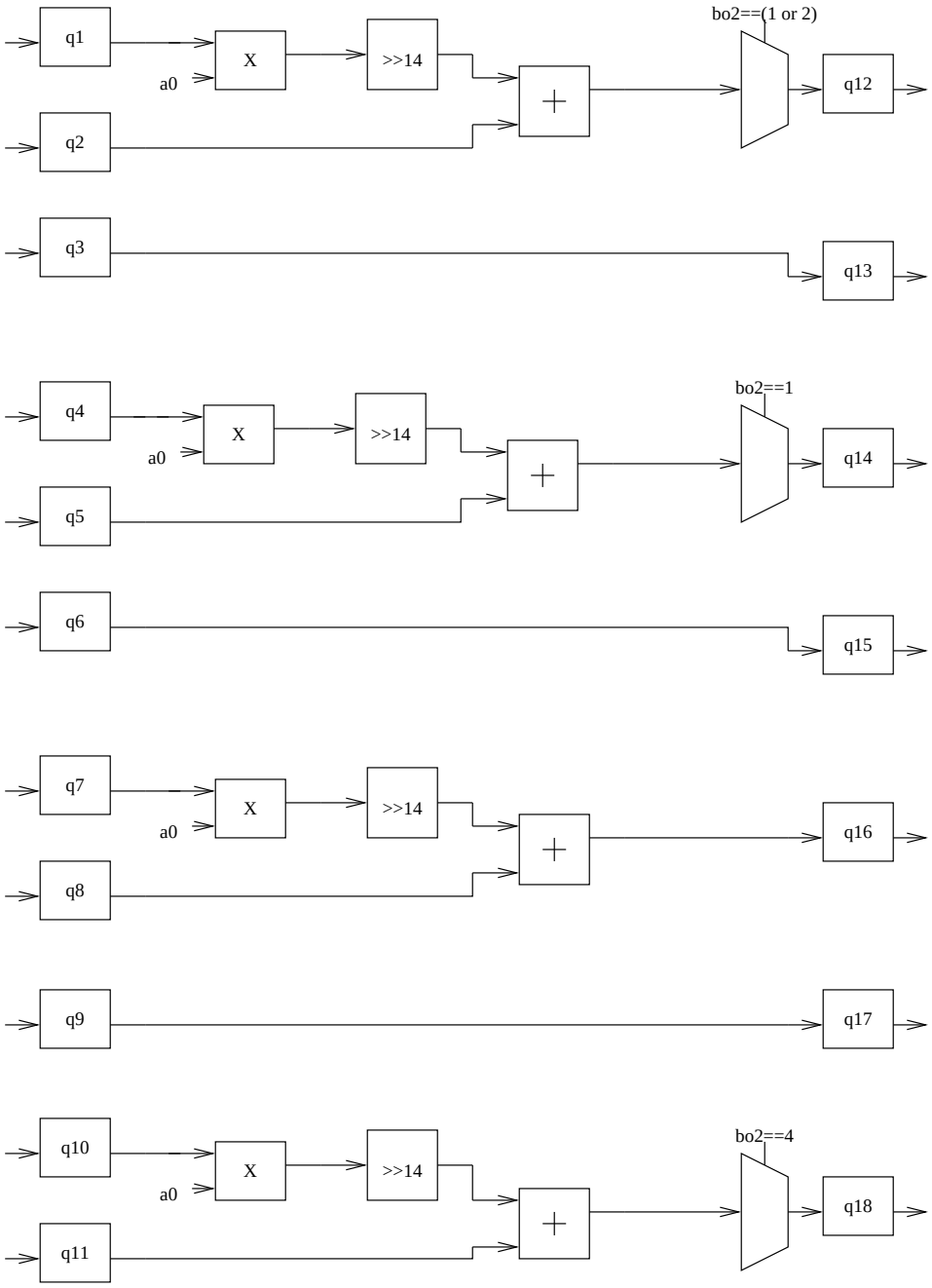

Figure 5.5: Second Pipeline stage in the DWT module

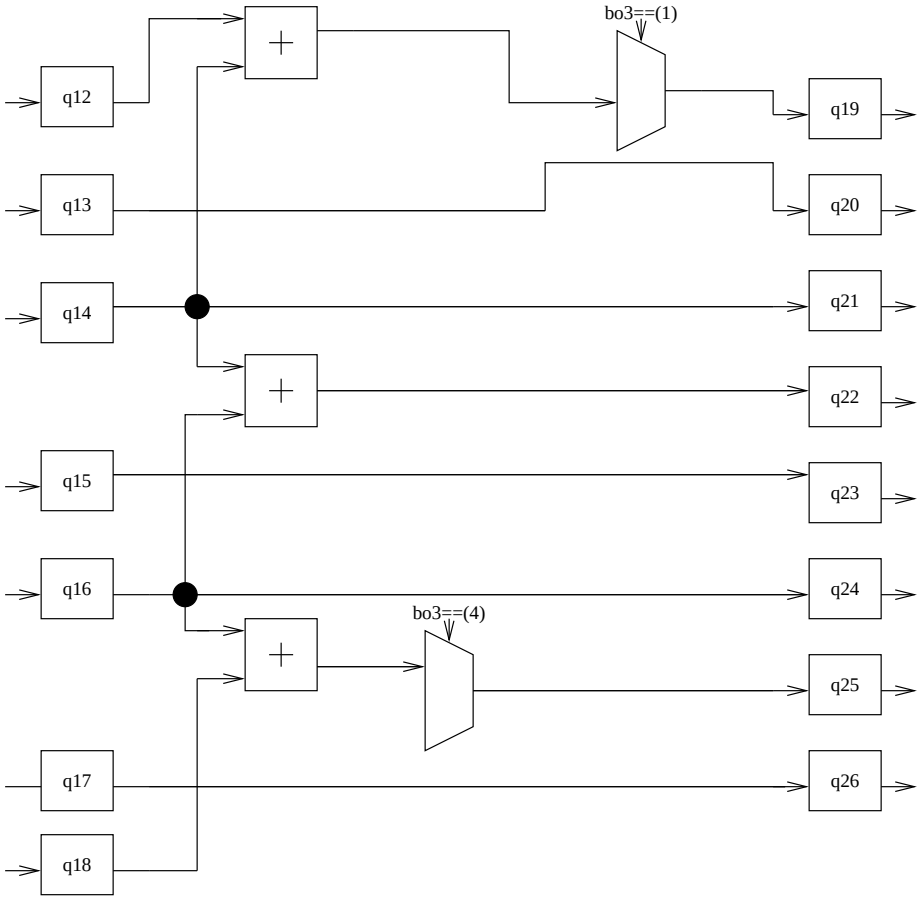

Figure 5.6: Third Pipeline stage in the DWT module

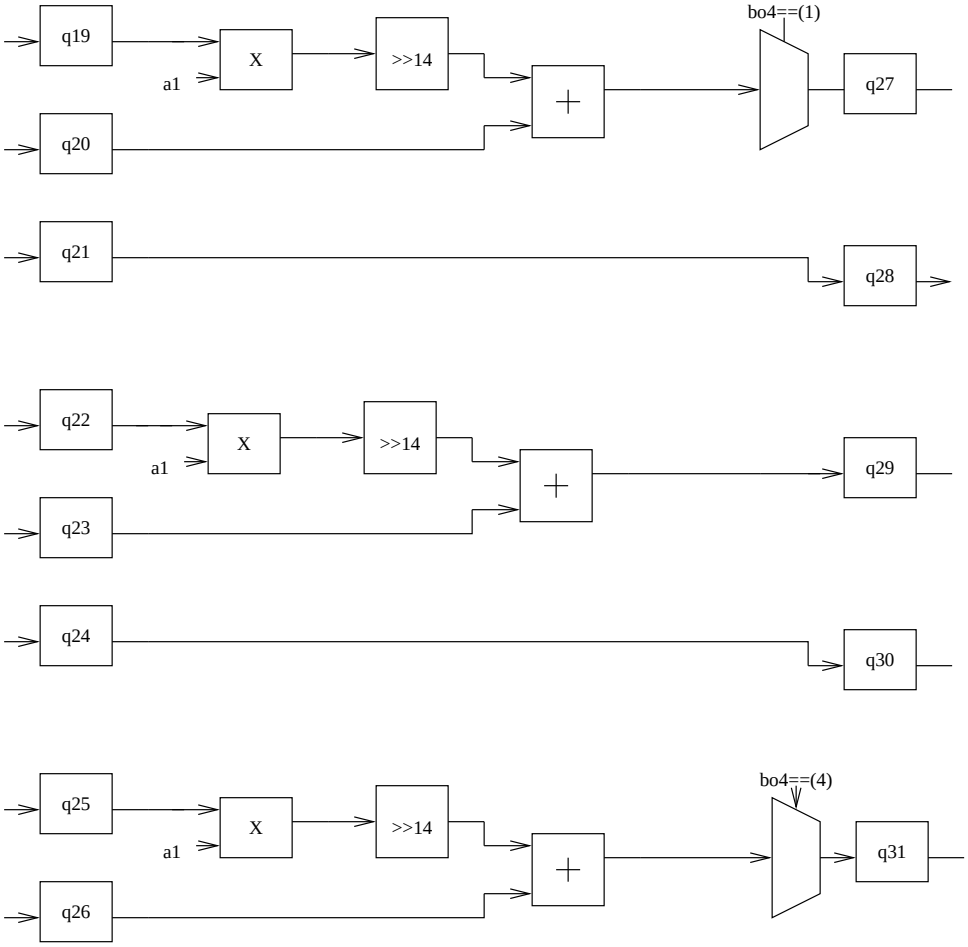

Figure 5.7: Fourth Pipeline stage in the DWT module

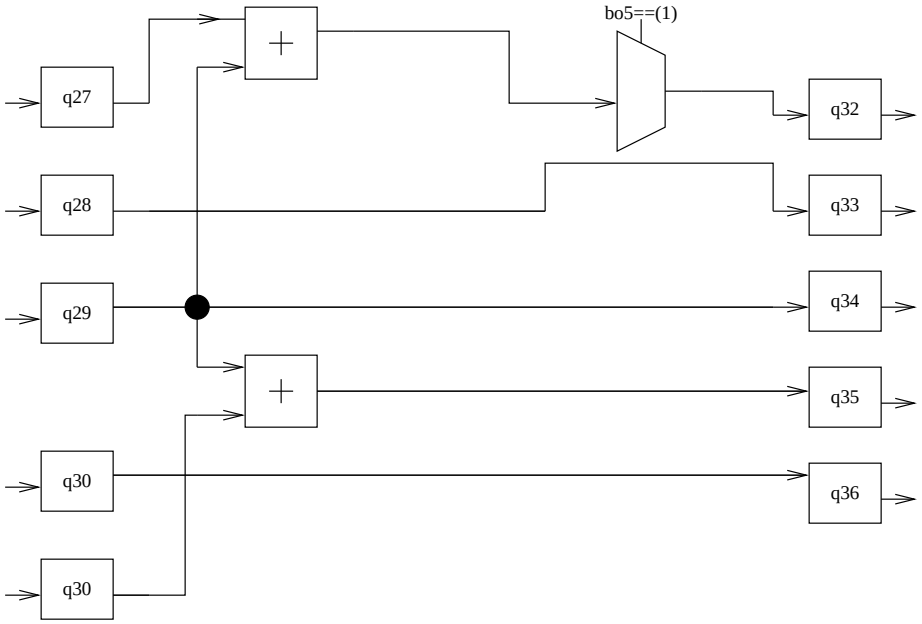

Figure 5.8: Fifth Pipeline stage in the DWT module

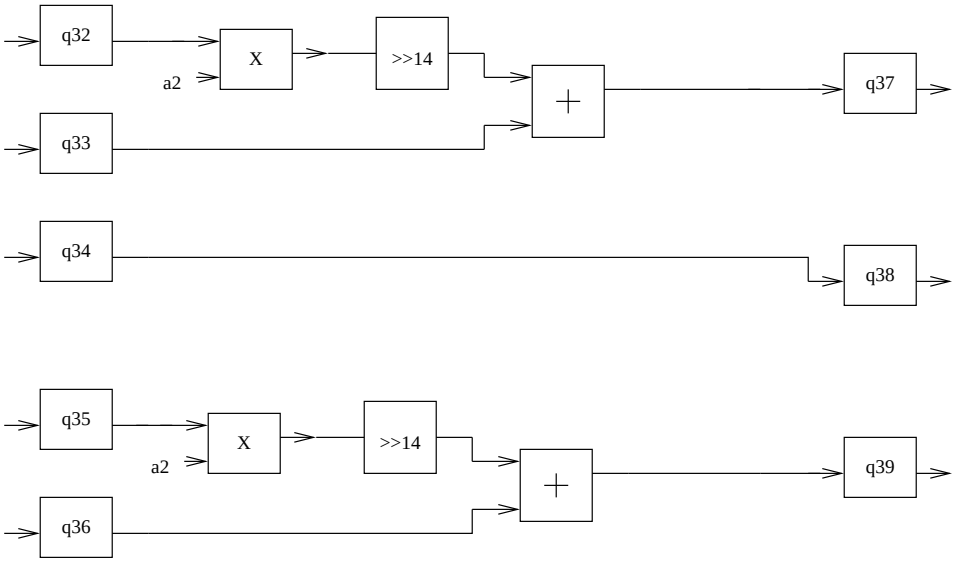

Figure 5.9: Sixth Pipeline stage in the DWT module

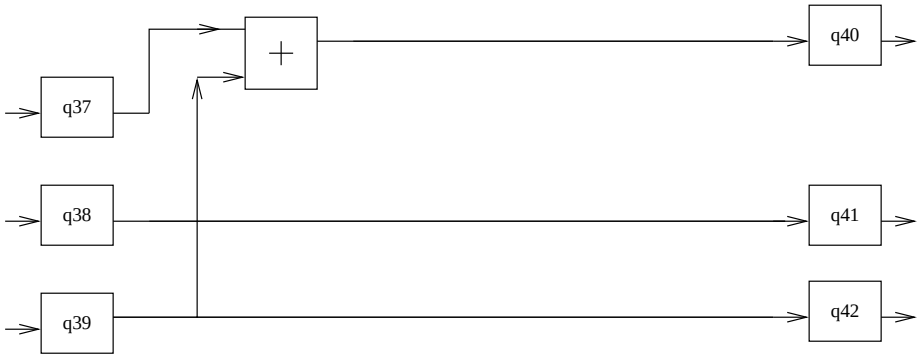

Figure 5.10: Seventh Pipeline stage in the DWT module

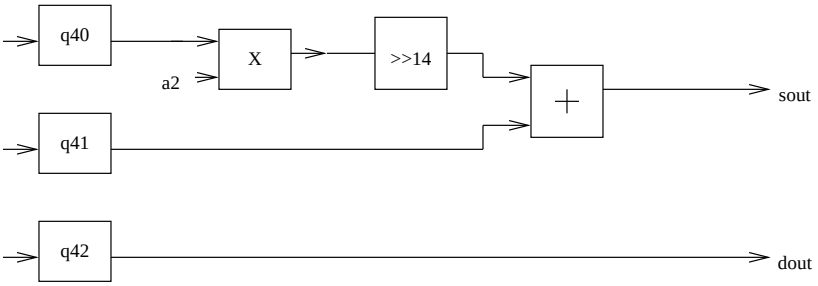

Figure 5.11: Output stage in the DWT module

the same time, two row IDWT modules were required to get back the coefficients at the same time for use by the column IDWT module.

5.3.1 Row Processing

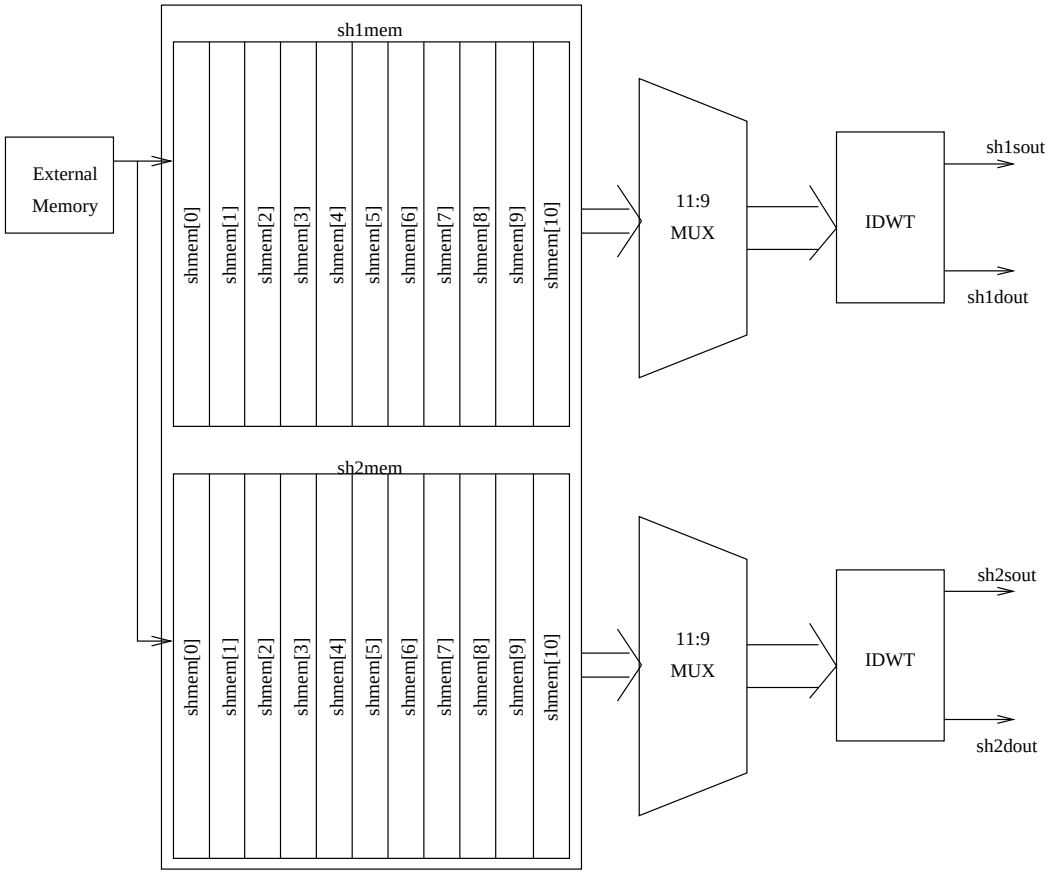

Figure 5.12: Block diagram of Row Processing part of IIWT hardware model

Figure 5.12 shows the block diagram of the row processing part of the IIWT hardware model. The row IDWT modules had the pipelined architecture designed in section 4.8 of the previous chapter. One of the modules took in coefficients from the LL and the LH bands to give two column IWT approximation coefficients while the other took in coefficients from HL and HH bands to give the two column IWT detail coefficients. Since each of the IDWT modules required 9 inputs at a time, two intermediate memories with 9 banks were

required. Eleven memory banks were used to avoid using two port expensive memories. Each memory bank had the size to store half the row length of the image as the row length of each subband is half of the image and were initialized to contain zeros.

In each row processing cycle, two data elements from the two columns in both LL and LH bands were read in and stored in the appropriate memory banks. Thus, each row computation required a total of 4 memory accesses, and hence 4 cycles were available to do the row computation. The boundary computations for the IDWT required a minimum of 6 data elements. Hence the valid row processing outputs were obtained after the first 3 columns of LL band and first 3 columns of LH bands were read into the last 6 memory banks of the first row IIWT. Each memory bank stored a column of the subbands. Since 4 outputs were produced after each computation after the initial latency cycles, the throughput of one output per clock cycle was maintained. The second row IIWT also did the same thing but used the HL and HH bands.

The Verilog and Matlab implementations used a finite state machine to generate the read enable signals for the appropriate intermediate memory banks. Similar to the implementation in DWT, the machine state indicated which memory bank contained the oldest row and which contained the most recent column. The value of the state changed when all the rows in a top to down scan were processed and two new columns were read into the intermediate memory bank. The state machine model used in the implementations is shown in the figure 5.13. Even though the FSM shown in the figure describes the read and write signals for $shmem1$, the same pattern was used for $shmem2$ also. The intermediate memory bank resembled a FIFO. It was observed from the state machine model the value of the state variable also indicated which memory bank contained the topmost values in the FIFO. Thus the state variable value was used by the multiplexers to find the 9 inputs to give to the column IDWT modules from the 11 inputs it received from the memory banks.

The outputs from each of the column IDWT modules were stored in a 9 element register array. One of the approximation coefficient along with one of the detail coefficient was stored in one register array, while the other two remaining coefficients were stored in the other register array.

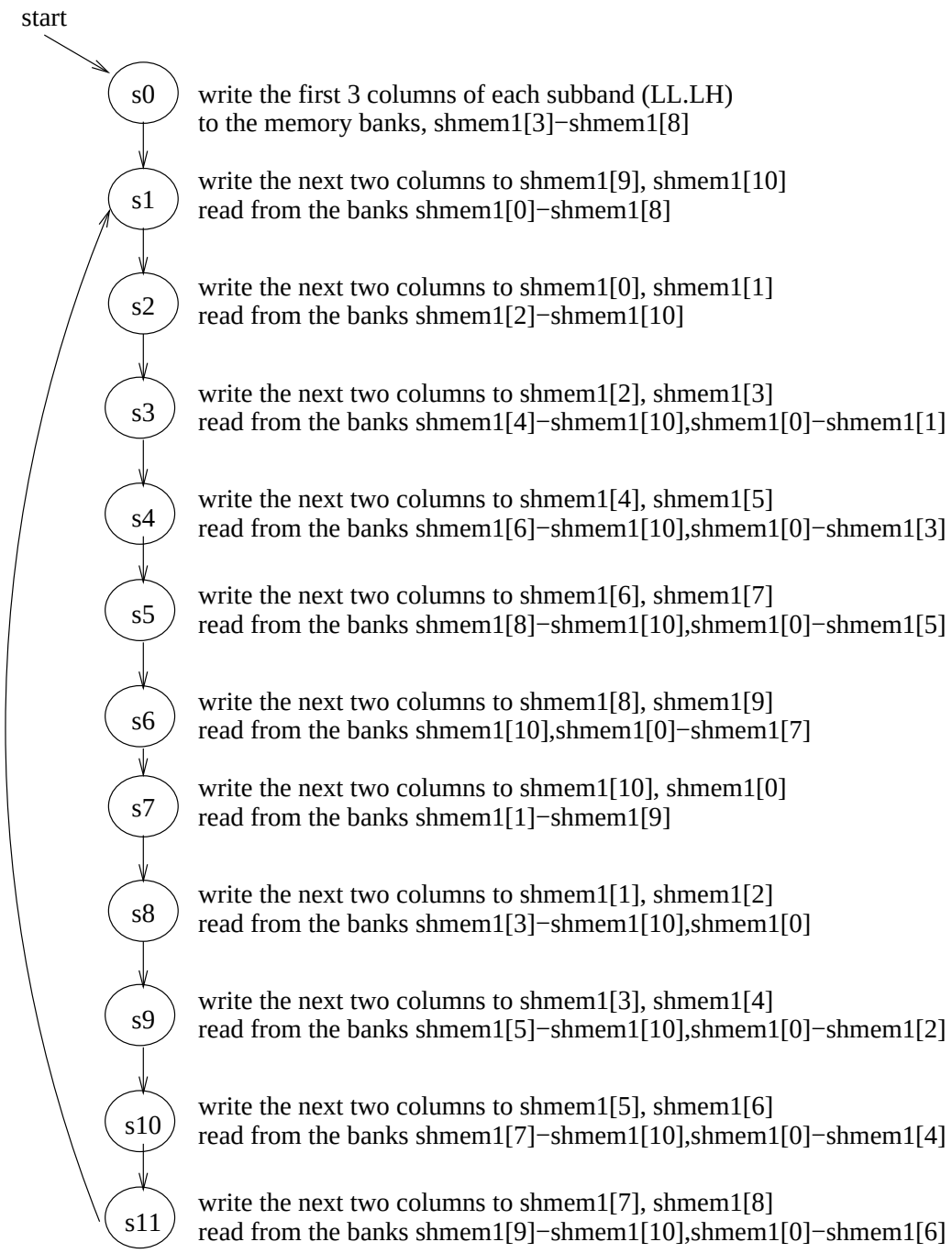

Figure 5.13: FSM used to generate read and write enable signals for the intermediate memory bank in the 2D-IIWT hardware

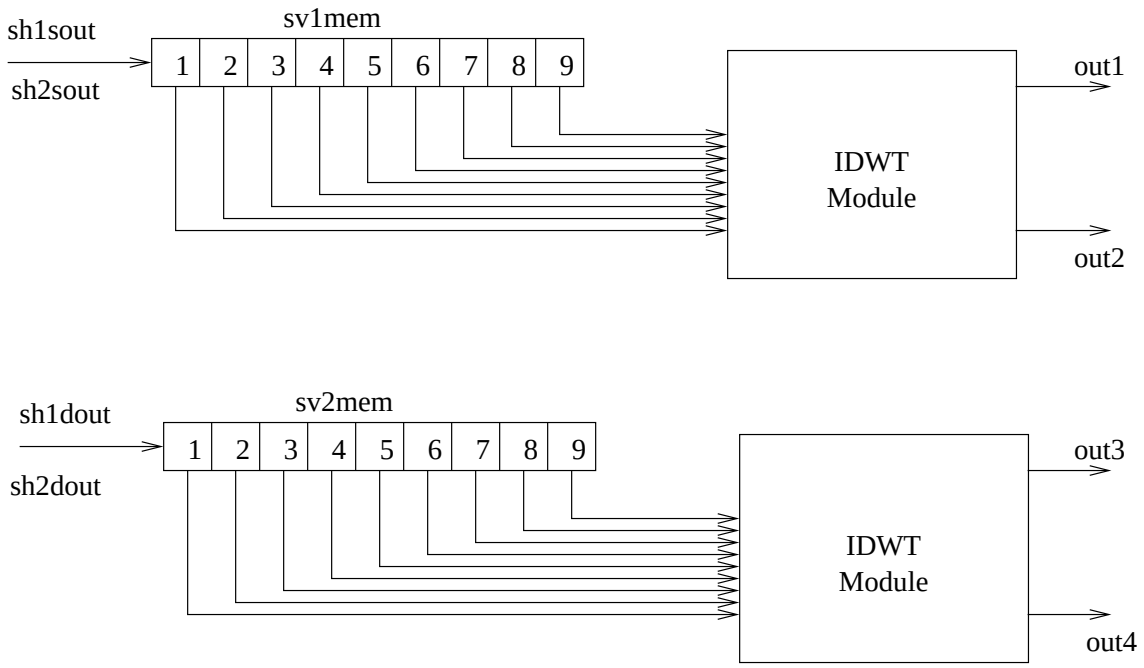

Figure 5.14: Block diagram of Column Processing part of 2D-IIWT hardware model

5.3.2 Column Processing

The column processing could start as soon as 3 valid computations in the row processing were over. The three valid computations in the row IDWT modules gave 6 coefficients which were stored in each of the register arrays. These register arrays were used to give the inputs to the column IDWT modules. The block diagram in 5.14 shows the column processing part of the 2D-IIWT model. The column IDWT module also worked at the same rate as the row IDWT giving 4 outputs after each computation with each computation using 4 clock cycles. The 4 cycle requirement for the computation was due to the row IDWT, because the column IDWT had to be in synchronization with the row IDWT to avoid over running the data in the register arrays.

Thus each of the column IDWT modules gave as outputs the even and odd columns of the original image, which were interleaved and stored into the external memory.

5.3.3 IDWT module

There were four instances of the IDWT module in the implementation of the inverse integer wavelet transform. The IDWT module used the data dependency graph based pipelined architecture design for the 1D-IIWT model explained in section 4.8. Each IDWT module required 20 inputs including the input clock to do the computations. The IDWT module gave two outputs. The twenty inputs to the IDWT module included the four wavelet lifting filter coefficients, six boundary indicators and the nine coefficients used in the computations. The IDWT modules used to do the horizontal IIWT received the nine coefficients from the external memory and the results of these computations were stored in buffers. The output of the row IDWT modules were the column IWT coefficients in even and odd positions. The outputs from the row IDWT modules were paired and stored in the even and odd positions in the buffers. Each buffer had space to store nine values and these buffer contents were given to the two IDWT modules used in the column IIWT computations .

It was shown in section 4.8 that in order to develop the pipelined architecture for the 1D-IIWT based on data dependency, seven cut set retimings were required. Each cut set retiming introduced a delay or a pipelined stage. All the registers except in the last two pipelined stages had to obey the boundary conditions and hence six registers were

needed for each stage to indicate the boundary conditions for that stage. While doing the horizontal inverse wavelet filtering, the data in the rows were read one after another. Thus the boundary conditions for the computations were the same for all rows in a top to bottom scan and changed only in the next top to bottom scan. However in the column inverse filtering, since the computations did not proceed to the next column before completing the current column, the boundary conditions repeated periodically ie., the conditions at the beginning of the column or at the end of the column repeated when the next column computations started.

In order to keep track of the boundary conditions, the row IDWT modules used six boundary indicator variables and the column IDWT modules used a different set of six boundary indicator variables. The six boundary indicators used in both the cases resembled an array of flip-flops with output value of a flip-flop being passed to next at the positive clock edge. Variables were used to keep track of the number of columns being read in the row IDWT modules and the number of rows used in the column IDWT computations. These variable values were used to set the correct values in the first flip-flop of the array ie., first boundary indicator variable. The figures 5.15, 5.16, 5.17, 5.18, 5.19 5.20, 5.21 and 5.22 show the eight pipelined stages in the IDWT module.

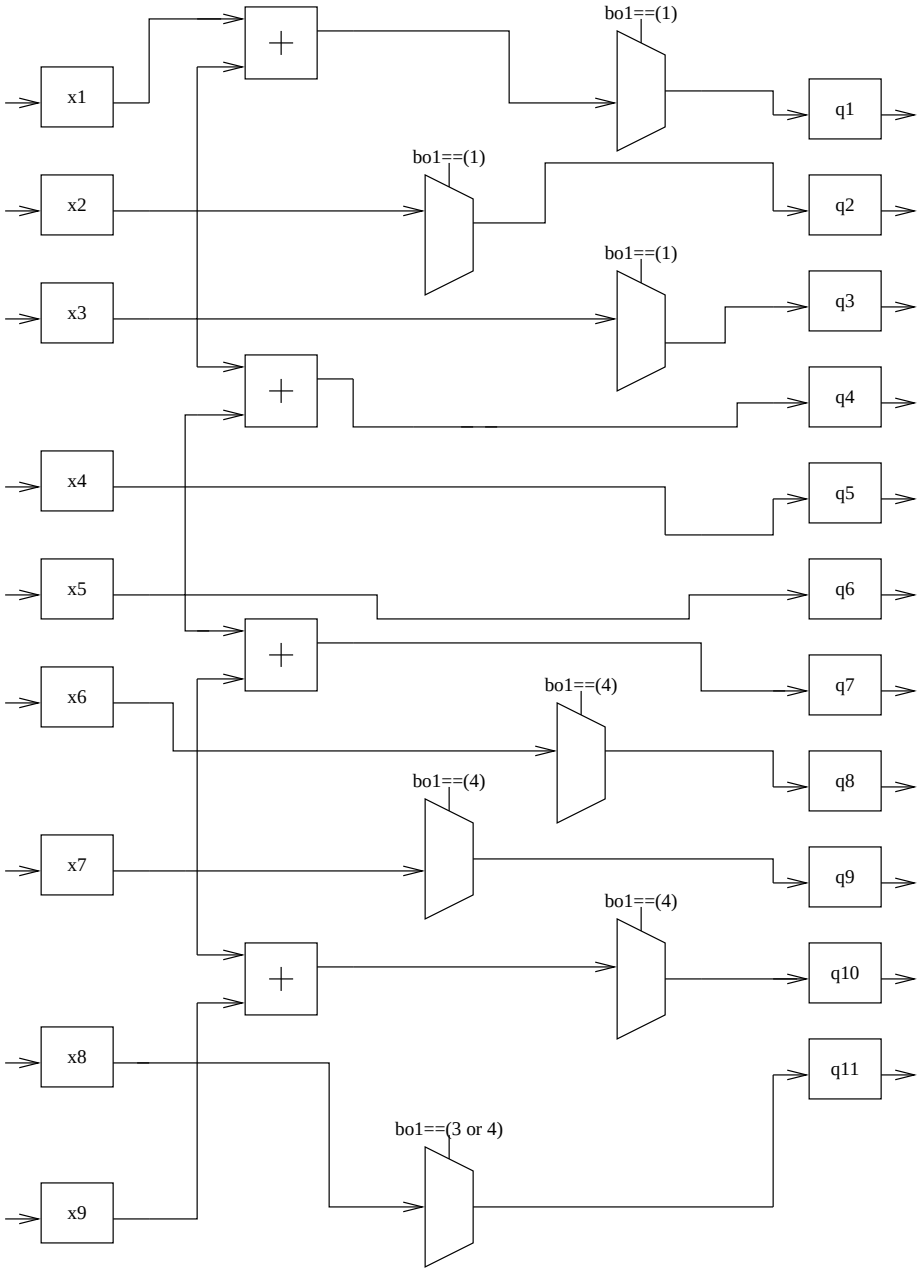

Figure 5.15: First Pipeline stage in the IDWT module

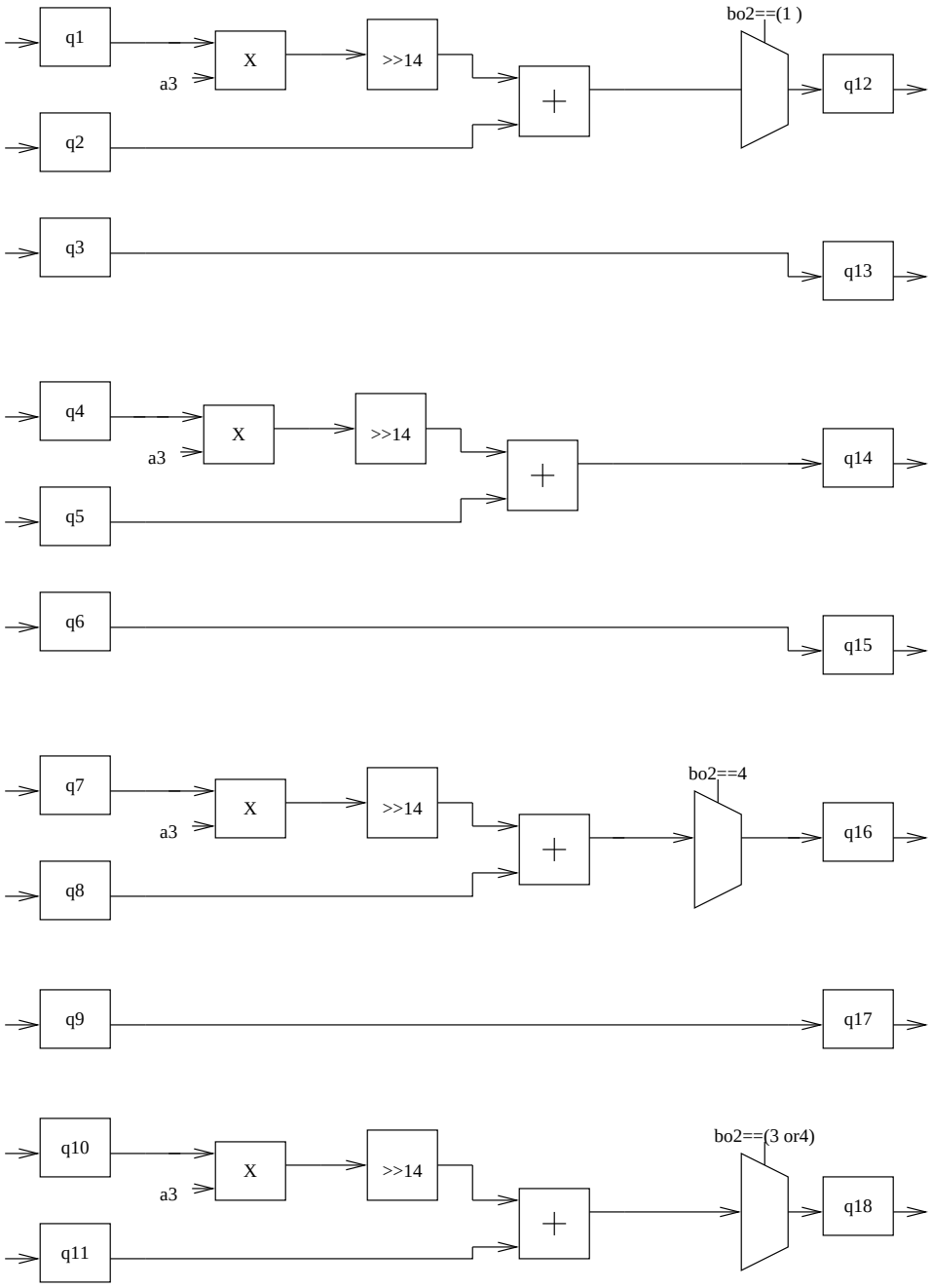

Figure 5.16: Second Pipeline stage in the IDWT module

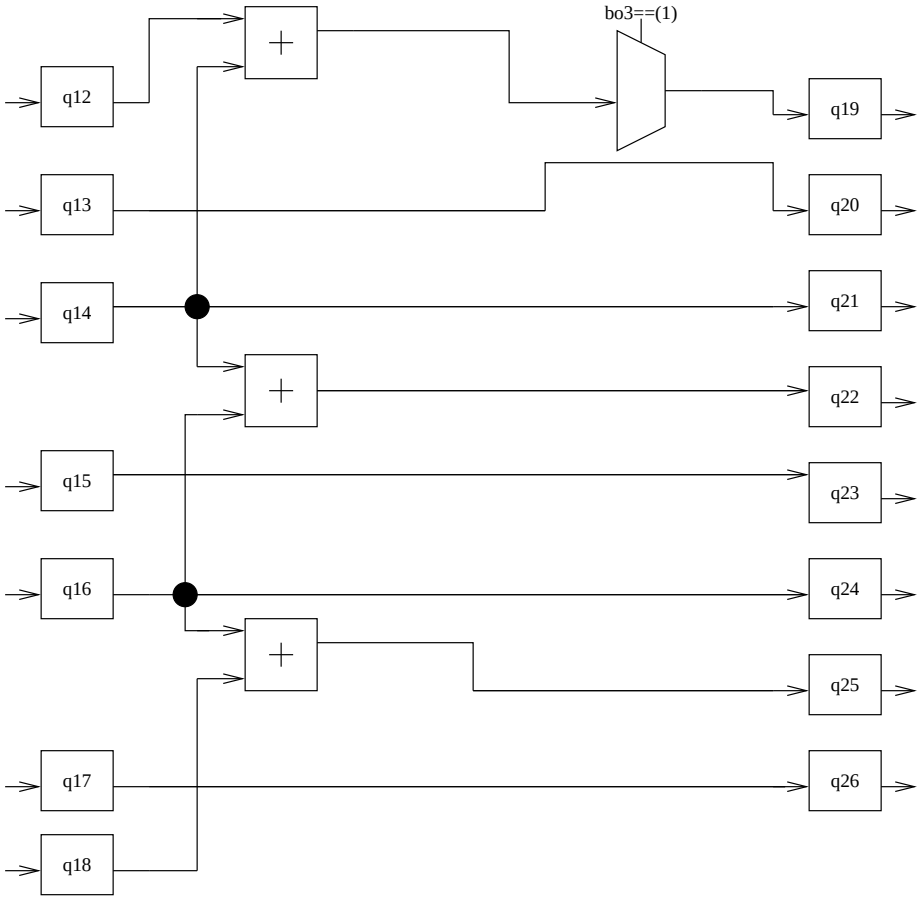

Figure 5.17: Third Pipeline stage in the IDWT module

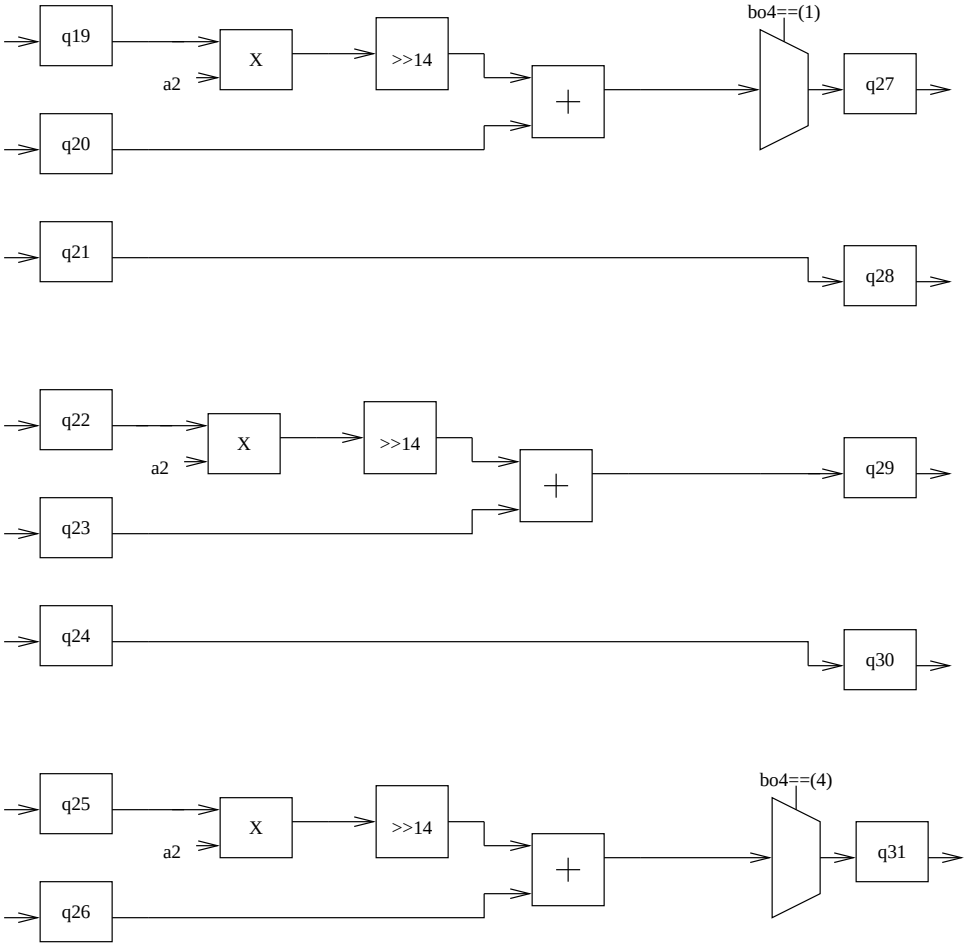

Figure 5.18: Fourth Pipeline stage in the IDWT module

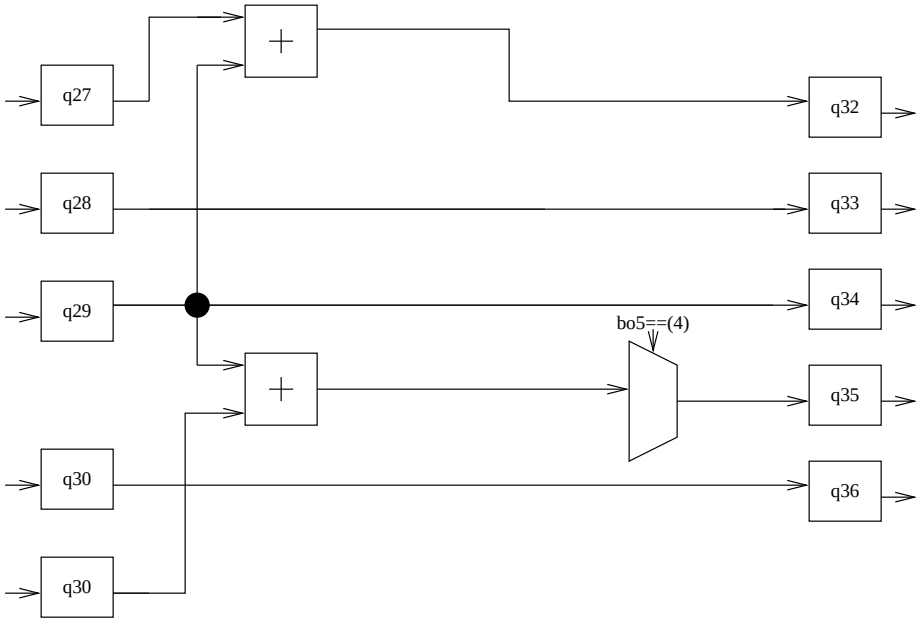

Figure 5.19: Fifth Pipeline stage in the IDWT module

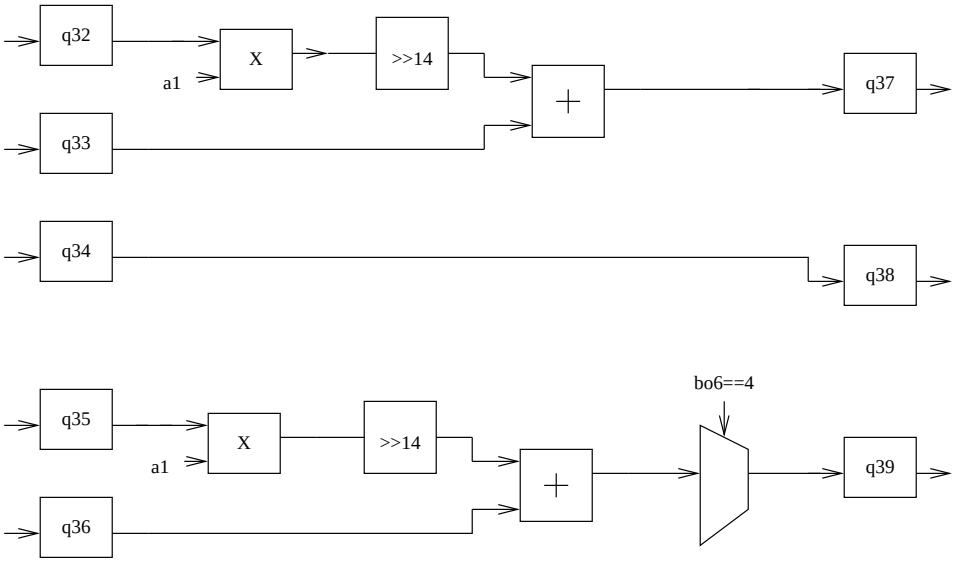

Figure 5.20: Sixth Pipeline stage in the IDWT module

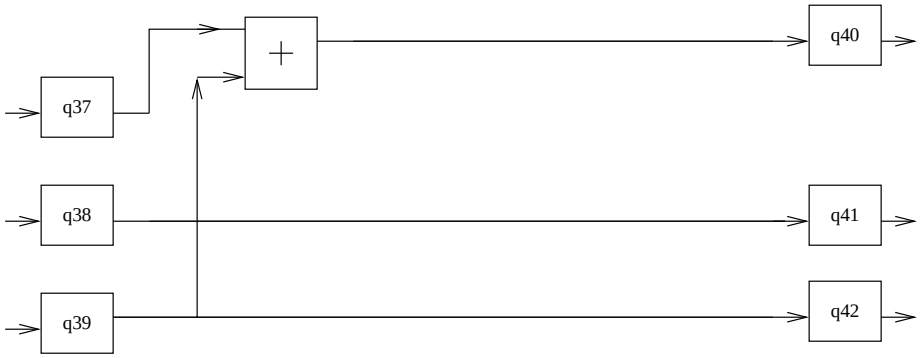

Figure 5.21: Seventh Pipeline stage in the IDWT module

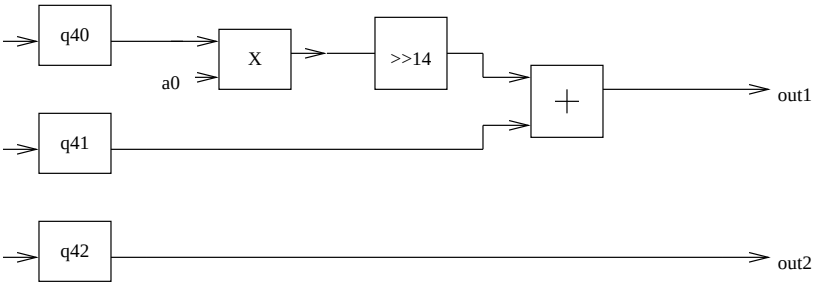

Figure 5.22: Output stage in the IDWT module

Chapter 6

Results

6.1 Introduction

This chapter tabulates the results obtained for one level of the 2D-Integer Wavelet Transform (IWT) decomposition and reconstruction of images using the hardware model designed in the previous chapter. The hardware implementation was done using Verilog and the results were compared with the Matlab implementation of the lifting scheme IWT.

6.2 Test Results

The Verilog implementation of the hardware model proposed in the previous chapter was done and tested using various 256 gray scale images. The images were obtained from USC-SIPI database [6]. The images were scaled down to a size of 128 x 128 for testing purposes. The results showed that there was no error while reconstructing the image. The tabulated test results are given below

The original images of couple.tiff, lena.gif and elaine.tiff (128X128) and their corresponding test outputs from Verilog after one level of 2D-IWT decomposition and reconstruction are shown in figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8 and 6.9.

Table 6.1: Maximum Error for images decomposed and reconstructed using the proposed hardware model

Sl No:	Image name	Image size	Maximum Error
1	couple.tiff	128x128	0
2	elaine.tiff	128x128	0
3	lena.gif	128x128	0

Figure 6.1: Original 128x128 couple.tiff

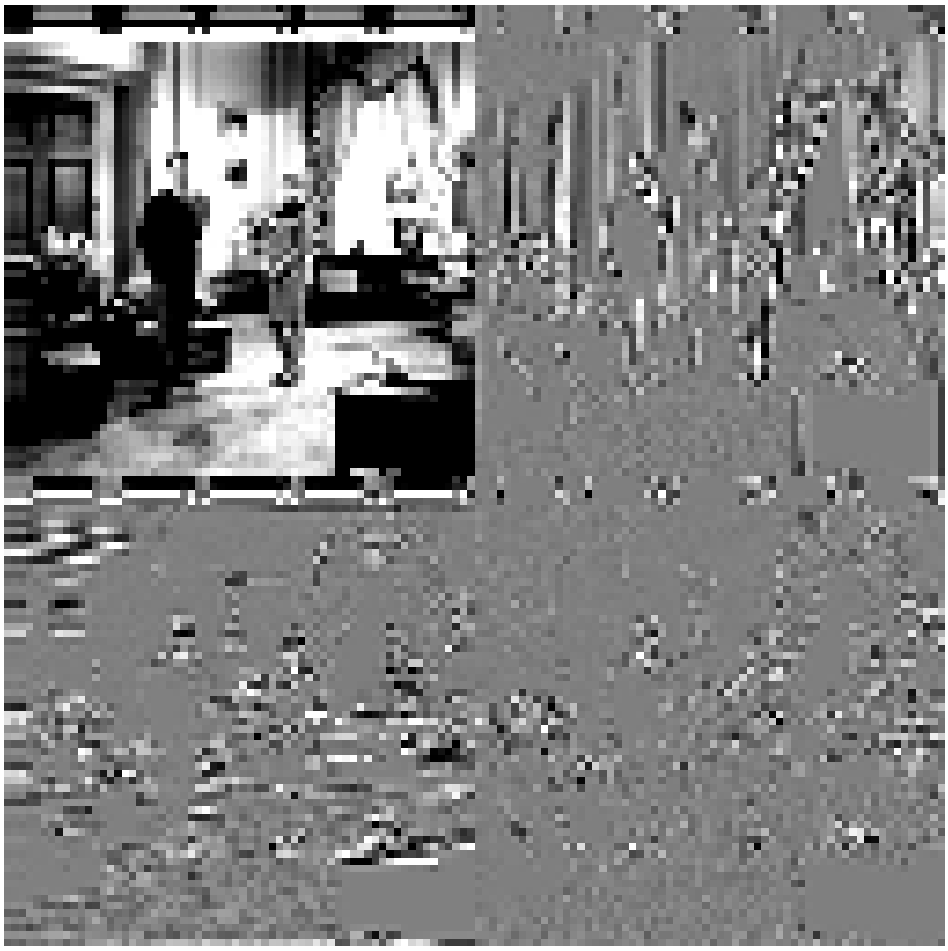

Figure 6.2: 2D-IWT decomposition of couple.tiff

Figure 6.3: Reconstructed image of couple.tiff

Figure 6.4: Original 128X128 lena.gif

Figure 6.5: 2D-IWT decomposition of lena.gif

Figure 6.6: Reconstructed image of lena.gif

Figure 6.7: Original 128X128 elaine.tiff

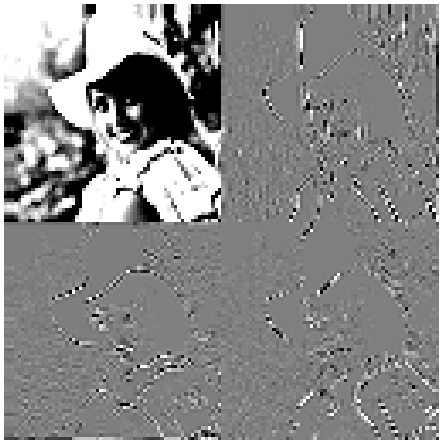

Figure 6.8: 2D-IWT decomposition of elaine.tiff

Figure 6.9: Reconstructed image of elaine.tiff

Chapter 7

Conclusions and Future Research

7.1 Introduction

This thesis presented the design and implementation of the hardware architecture for the 2D-integer wavelet transform of images for use in region of interest coding in telemedicine applications. The first portion of the thesis focussed on the background information and the problem statement in the design of an efficient two dimensional wavelet lifting architecture. The latter portion of the thesis covered the design techniques used and the results obtained after the implementation of the proposed hardware architecture. The contributions made by this thesis are summarized below.

7.2 2D-IWT Architecture Design and Implementation

As mentioned in the problem statement for the thesis, one of the major requirements for the image transforms used in telemedicine is the ability to reconstruct the image with little or no loss. In addition, the two major problem areas in the development of the two dimensional wavelet transform architecture is the memory access time and the data communications between the parallel processors.

W. Sweldens and Daubechies proposed the integer wavelet transform concept [5]. This concept was used in this thesis to overcome the losses usually incurred in the recon-

struction phase of the image transforms. The round off errors introduced by floating point to fixed point computation conversions were reduced by using scaling, rounding and sign extension.

The memory access time in the 2D-IWT architecture results in idle hardware and low computational efficiency. This was avoided by the use of the two dimensional separable transform which handled row and column filtering at the same time. The column processing was done first followed by the row processings ince the memory savings in this case was around one half that of the case where row processing was done first followed by column processing. The lifting architectures for the 1D-IWT found in the literature showed that an increased amount of data communications was required between the parallel processors if parallel processing was used.

This thesis investigated the possibility of using a pipelined approach to serial processing for the 1D-IWT lifting architecture and found that when used in the 2D-IWT the throughput was very low at the rate of one output per twenty clock cycles. This led to a different and more efficient architecture based on the data dependency graph of the lifting structure for the 1D-IWT which gave a throughput of one output per clock cycle for the 2D-IWT if we ignored the initial latency period of fourteen clock cycles due to the use of pipelining.

The architecture was tested by using several images and gave perfect reconstruction after 2D-IWT decomposition.

7.3 Future Research

Even though this thesis contributed a part of the hardware implementation of the region of interest coding for use in Telemedicine applications, further research work is needed to complete the implementation.

- The wavelet coefficients obtained after the required levels of decomposition have to be partitioned into subsets using the set partioning and rearrangement technique mentioned in 1.2. This can be done in hardware by appropriate calculation of the address to which the final decomposed data is written.

- The subsets obtained after partitioning have to be quantized such that bit planes corresponding to each subset is produced. The bit planes at the same level for each

subset represent a layer of data. The implementation of succesive quantization would be more efficient in hardware due to the natural bit wise representation used in hardware.

- The run length encoder encodes the bits in each layer resulting in a compressed bit stream. The run length encoder could be implemented by means of a state machine which counts the number of bits in the input and gives out the encoded words.
- The receiver side would have the inverse functionality of each block implemented in the sender side.

Bibliography

[1] Michael D. Adams and Rabab Ward. Wavelet transforms in JPEG 2000 standard. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1749–1752. IEEE, May 2001.

[2] Winser E. Alexander. Digital signal processing algorithms and architecture. ECE 747 Course Notes, North Carolina State University, Raleigh, NC, 2003.

[3] Kishore Andra, Chaitali Chakrabarti, and Tinku Acharya. A VLSI architecture for lifting based forward and inverse wavelet transform. *IEEE transactions on Signal Processing*, 50(4):966–977, April 2002.

[4] S. Barua, J.E. Carletta, K.A. Kotteri, and A.E. Bell. An efficient architecture for lifting-based two dimensional discrete wavelet transforms. In *Proceedings of the 14th ACM Great Lakes symposium*, pages 61–66. ACM Press, 2004.

[5] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo. Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis*, 5(3):332–369, 1998.

[6] USC-SIPI Image Database. USC-SIPI image database. Available from URL: http://sipi.usc.edu/services/database/Database.html.

[7] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl*, 4(3):245–267, 1998.

[8] Toujd Ebrahami, Mathias Larsson, Joel Askelof, and Charilaos Cristopaulos. ROI coding in JPEG 2000 for interactive client/ server applications. In *Proc. of the IEEE Third*

Workshop on Multimedia Signal Processing (MMSP), pages 389–394. IEEE, September 1999.

[9] S. Gnavi, M. Grangetto, E. Magli, and G. Olmo. Wavelet kernals on aDSP: A comparison between lifting and filter banks for image coding. *EURASIP Journal on Applied Signal Processing*, 2002(9):981–989, April 2002.

[10] Karen L. Gray. The JPEG 2000 tutorial. Available from URL: http://www.lkn.ei.tum.de/studium/mmprog/jpeg2000/content.pdf.

[11] Chao-Tsung Huang, Po-Chih Tseng, and Liang-Gee Chen. Efficient VLSI architecture of lifting based discrete wavelet transform by systematic design method. In *IEEE International Symposium on Circuits and Systems, ISCAS 2002*, volume 5, pages 565–568. IEEE, May 2002.

[12] Wenqing Jiang and Antonio Ortega. Lifting factorization based discrete wavelet transform architecture design. *IEEE transactions on Circuits and Systems for Video Technology*, 11(5):651–657, May 2001.

[13] Zervas N.D., Anagnostopoulos G.P., Spiliotopoulos V., Andreopoulos Y., and Goutis C.E. Evaluation of design alternatives for the 2D-discrete wavelet transform. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(12):1246–1262, December 2001.

[14] Majid Rabbani. The JPEG 2000 still image compression standard. Available from URL: http://jj2000.epfl.ch/jjpublications/papers/011.pdf.

[15] Amir Said and William A. Pearlman. A new, fast and efficient image codec based on set partitioning in hierarchial trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[16] Po-Chih Tseng, Chao-Tsung Huang, and Liang Gee Chen. Generic RAM based architecture for two dimensional discrete wavelet transform with line based method. In *Asia-Pacific Conference on Circuits and Systems, APCCAS '02*, volume 1, pages 363–366, October 2002.

[17] Bryan Usevitch. A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000. *IEEE Signal Processing Magazine*, 18(5):22–35, September 2001.

[18] M. Vishwanath. The recursive pyramid algorithm for the discrete wavelet transform. *IEEE Transactions on Signal Processing*, 42(3):673–676, March 1994.

[19] Po-Cheng Wu and Liang-Gee Chen. An efficient architecture for two-dimensional wavelet transform. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(4):536–545, April 2001.

[20] S. H. Yoon, Ji Hyun Lee, J. H. Kim, and W. E. Alexander. Medical image compression using post segmentation approach. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, volume 5, pages 609–612, May 2004.