1999

# A 2D DWT architecture suitable for the Embedded Zerotree Wavelet Algorithm

James Martinez

*Edith Cowan University*

# A 2D DWT ARCHITECTURE SUITABLE FOR THE EMBEDDED ZEROTREE WAVELET ALGORITHM

A Thesis Submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering

**James Martinez**

November 1999

Faculty of Health and Sciences

School of Engineering and Mathematics

Edith Cowan University

Western Australia

## ABSTRACT

Digital Imaging has had an enormous impact on industrial applications such as the Internet and video-phone systems. However, demand for industrial applications is growing enormously. In particular, internet application users are growing at a near exponential rate

The sharp increase in applications using digital images has caused much emphasis on the fields of image coding, storage, processing and communications. New techniques are continuously developed with the main aim of increasing efficiency. Image coding is in particular a field of great commercial interest. A digital image requires a large amount of data to be created. This large amount of data causes many problems when storing, transmitting or processing the image. Reducing the amount of data that can be used to represent an image is the main objective of image coding.

Since the main objective is to reduce the amount of data that represents an image, various techniques have been developed and are continuously developed to increase efficiency. The JPEG image coding standard has enjoyed widespread acceptance, and the industry continues to explore its various implementation issues. However, recent research indicates multiresolution based image coding is a far superior alternative.

A recent development in the field of image coding is the use of Embedded Zerotree Wavelet (EZW) as the technique to achieve image compression. One of The aims of this theses is to explain how this technique is superior to other current coding standards. It will be seen that an essential part of this method of image coding is the use of multiresolution analysis, a subband system whereby the subbands are logarithmically spaced in frequency and represent an octave band decomposition. The block structure that implements this function is termed the two dimensional Discrete Wavelet Transform (2D-DWT).

The 2D DWT is achieved by several architectures and these are analysed in order to choose the best suitable architecture for the EZW coder. Finally, this architecture is implemented and verified using the Synopsys Behavioural Compiler and recommendations are made based on experimental findings.

## DECLARATION

I certify that the attached project is my own work and that any material drawn from other sources has been acknowledged.

## ACKNOWLEDGEMENT

I would like to thank Dr Hon Nin Cheung and Ken Ang Li-Minn who assisted me in the completion of this project. Also I would like to thank Cole Bergensen for all help given with the Synopsys VHDL software.

Date
11-7-99

# TABLE OF CONTENTS

Abstract ii
Declaration and Acknowledgement iii

**Introduction** 1

**CHAPTER 1**
**WAVELET AND IMAGE CODING**

1.0 Basic Wavelet theory 5
1.1 Choice of wavelet basis 7
1.2 Dilation 9
1.3 Image Compression 10
1.4 Subband transforms 11
1.5 Transform properties 12
1.51 Scale and orientation 13
1.52 Spatial localisation 13
1.53 Orthogonality 14
1.6 Some Transforms 15
1.61 The Gabor Transform 15
1.62 The DCT Transform 16
1.63 The Laplacian pyramid 17
1.64 Quadrature mirror filters 18
1.65 Asymmetrical QMF filters 19
1.66 Non-Separable QMF transforms 19
1.7 Multiresolution Analysis 20
1.8 Wavelet transform Vs Subband decomposition 22
1.9 Wavelet properties 23
1.10 Entropy 26

**CHAPTER 2**
**IMAGE CODERS**

2.1 Basic image coders 27
2.2 Entropy coding and arithmetic coding 28
2.3 Transform coding 28
2.4 Run length coding 29
2.5 Zerotree coding 29
2.6 Embedded zerotree wavelet transform 30

**CHAPTER 3**
**SEARCH STRATEGIES AND EZW COMPONENTS**

3.1 Tree search strategies 34
3.2 Depth first search and Breadth first search 35
3.3 EZW components 36
3.31 Discrete Wavelet transform 37
3.32 Mallat's tree algorithm 38
3.4 Two dimensional discrete wavelet transform (2D-DWT) 39
3.5 Quantization zerotree 40
3.6 Arithmetic coding 40

**CHAPTER 4**
**TWO DIMENSIONAL DWT ARCHITECTURES**

4.1 Two dimensiona DWT architectures 41
4.2 Comoputational complexity of the DWT 42
4.3 Data dependencies within DWT 43
4.4 Finite precision effect 45
4.6 Image quality 46
4.61 Subjective and objective image quality 46

**CHAPTER 5** /
**PROPERTIES OF DWT ARCHITECTURES**

5.0 Architecture considerations 48
5.1 Systolic architecture 48
5.2 Lattice filter type architecture 50
5.3 Pararell architecture 52
5.4 2D-DWT Processor type architecture 54
5.5 Recursive type architecture 57
5.6 Recommended architecture 58
5.7 XILINK XC40000 61
5.8 MAC implementation 62
5.9 FIR filter implementation 63
5.10 Memory bank 64
5.11 Control unit 66

CHAPTER 6
VHDL IMPLEMENTATION

6.0 VHDL for hardware design 67
6.1 Hardware for implementation 69
6.2 Control 70
6.3 VHDL architecture implementation 77
6.4 Behavioural VHDL core 79
6.5 Filter variations 84
6.6 Structural architecture 85
6.7 Modularity 86
6.7 Structural computational unit 86
6.71 Structural multiplier 87
6.72 Structural adder 88
6.73 Structural multiplexer 89
6.74 Structural delay unit 89
6.75 Strutural memory bank 89
6.76 Structural control 90

CHAPTER 7
RESULTS

7.0 Introduction 91
7.1 Results and observations 92
7.2 Outcomes 94
7.3 Throughput 96
7.4 Latency 97
7.5 Comparison to other architectures 98
7.6 Input output considerations 99

CHAPTER 8
CONCLUSION

8.0 Conclusion 100
8.1 Further work 104
8.2 Concluding statement 104

**Bibliography** 105
Appendices 109

## INTRODUCTION

Digital images are used in many applications such as the internet and high definition TV (HDTV). An image can be considered as a positive function on a plane. The value of this function at each point specifies the luminance or brightness of the picture at that point. Digital images are sampled versions of such functions where the value of the function is specified only at discrete locations on the image plane, known as pixels. The standard representation of a digital image is then that of samples (pixels) residing on a rectangular lattice or matrix as shown in figure 1.0.

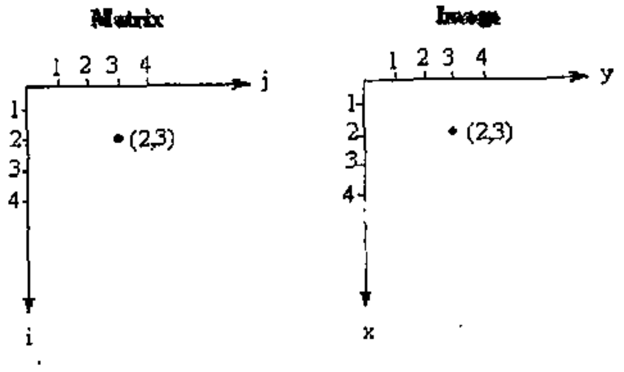


Fig. 1.0 Image representation as a matrix of luminance values

If an image is made of pixels of certain brightness placed in a rectangular matrix of size MxN, then there would be M xN pixels. Furthermore each pixel has a certain level of brightness which is represented to a pre-defined precision B (usually 8 bits), hence an image is composed of MxNxB bits. If an image is composed of 512 pixels and each pixel has a value of 0-255 for brightness which requires 8 bits, then the canonical representation of this image requires $512^2$ x 8 = 2097152 bits.

The huge amount of data required to represent an image is further aggravated when we consider that colour images requires even more data due to the representation of colour by a three dimensional vector function on a plane. Also most digital images have in excess of 512x512 pixels and hence require much larger storage space. Image coding consists of mapping images to strings of binary digits. The function of all image coders is to produce a binary string whose length is smaller the original canonical representation of the image by transforming the image.

In most cases, an image has a large amount of redundant data which is not required to represent the image accurately. In other words this redundant data can be removed without affecting the image quality significantly. Transform coding hence breaks the pixel array representing the image into a statistically uncorrelated data set . In plain language this means that the image is split into bands of different detail levels. When an image transformer is used effectively with a quantitiser, good compression can be achieved.

Quantizing a group of pixels together is known as vector quantization, or VQ. It has been shown [1] that in principle this method can achieve the highest compression that can be achieved by any coder. However it is also true that that the computational cost and delays experienced by these coders grow with dimensionally, limiting the practicality of VQ. For this reason and other difficulties, most practical coding algorithms have turned to transform coding instead of high dimensional VQ.

Transform coding consists of scalar quantization in conjunction with a linear transform to capture much the benefits of VQ without many of the problems experienced by VQ. Transform coding is successful if the basis functions of the transform represent the features of the signal accurately. At present, one of the most successful representations is the wavelet transform which can be viewed as a special case of a subband transform.

The theory underlying wavelets brings to bear a fundamentally different perspective than the frequency-based subband framework. The temporal properties of the wavelet transform have proved particularly useful in motivating some of the most recent coders. The Development of Embedded Zerotree Wavelet coding [2] motivated a flurry of activity in the area of zerotree wavelet algorithms. The inherent simplicity of the zerotree data structure, its computational advantages, as well as the potential for generating an embedded bitstream are all desirable qualities of good image coders.

This technique uses a two dimensional wavelet transform to break the data into low frequencies containing most of the image details and high frequencies containing most of the redundant data. It will be seen in this theses that the EZW algorithm achieves very good rate distortion performance and have a successive refinement property meaning that it generates an embedded code.

The purpose of this thesis is focused on investigating the advantages of the EZW algorithm over other existing compression techniques. A model of the EZW coder will also be presented and this will be discussed in detail. An essential functional block of the EZW coder is the 2-D Discrete Wavelet Transform and the aim of this theses is to investigate current efficient architectures to implement this function. A suitable architecture will be selected and then implemented and verified in VHDL using the Synopsys Behavioural Compiler or Accolade PeakVHDL Tools.

The organisation of this thesis is as follows. Chapter 1 provides an overview of image compression fundamentals and wavelets and provides a solid background to many fundamental issues. Chapter 2 discusses the EZW coder and compares this to other existing coders. Chapter 3 looks at the current EZW model and elaborates on the building blocks of this model. Chapter 4 discusses current 2-D DWT architectures and outlines a recommended architecture. Chapter 5 discussess VHDL modelling of the proposed architecture. Chapter 6 investigates and evaluates results of the VHDL simulations. Finally, Chapter 7 gives conclusions and further work for the overall research.

# CHAPTER 1

# WAVELETS AND IMAGE CODING

## 1.0 BASIC WAVELET THEORY

Wavelets are functions that can be used to filter other functions into low frequency components and multiple levels of high frequency components. The low frequency information is obtained from a scaling filter, and the high frequency information is obtained from detail, or wavelet, filters. The wavelet transform has an advantage over the Fourier transform in that it is able to deal with low frequency information over a large time and high frequency information over a short time. This makes wavelets desirable when dealing with functions containing a wide range of frequencies.

The Discrete Wavelet Transform (DWT) as shown in Eq.(1) is quite simple to compute once the basis functions have been chosen.

$$C_i = \sum_{J=0}^{N-1} b_{ij}\, a_j \tag{1}$$

Where:

$C_i$ = $i^{th}$ Transformed coefficient.
$b_{ij}$ = $j^{th}$ value of $i^{th}$ normalised basis functions.
$a_j$ = value of datum.
N = number of taps in the basis functions.

Similarly, the inverse transform is shown in Eq.(2).

$$\tilde{a}_i = \sum_{j=0}^{N-1} b_{ji} C_j \qquad (2)$$

Where:

$C_j$ = $j^{th}$ transformed coefficient.
$b_{ji}$ = $i^{th}$ value of $j^{th}$ basis function.
$\tilde{a}_i$ = reconstructed datum value.
N = number of taps in the basis functions.

The DWT is a one dimensional transform. Multi-dimensional data can be transformed by applying the transform to each dimension separately. Also, wavelets can be easily applied to different block sizes of the data. The basic wavelet function called the mother wavelet needs only to be translated and scaled to change the extend of the transform.

The Mallat algorithm for DWT [3] is a computationally efficient method of implementing the wavelet transform. The algorithm operates on a finite set of N input data, where N is a power of two. The data is passed through two convolution functions, each of which creates an output stream that is half the length of the original input. It will be seen later that the convolution functions are a low pass and a high pass FIR filters.

Furthermore the DWT is a convolution function. The property of convolution functions is that the low pass filter output contains most of the "information content" of the original input signal while the high pass filter output contains the difference between the true input and the value of the reconstructed input if it were to be reconstructed from only the information given in filter's output. In general, higher order wavelets tend to put more information in the low pass filter output and less in the high pass filter output.

The property of putting more information in the low pass filter output is very significant in image compression, if the average amplitude of the high pass filter is low enough, then the high pass filter output may be discarded without greatly affecting the quality of the reconstructed signal. An important objective of this thesis is to find wavelet functions which cause the high pass output to be nearly zero.

## 1.1 CHOICE OF WAVELET BASIS

Deciding on the optimal wavelet basis to use for image coding is a difficult problem. A number of design criteria, including smoothness, accuracy of approximation, size support, and filter selectivity are known to be important [4].

The simplest form of wavelet basis for images is a separable basis. This basis is formed from translations and dilations of products of one dimensional wavelets. Using separable transforms resolves the problem of designing efficient wavelets to a one dimensional problem, hence almost all current coders employ separable transforms. However recent work by Sweldens and Kovacevic [5] simplifies current difficulties with non-separable bases, and such bases may prove more efficient than separable transforms.

The prototype basis functions for separable transforms are $\phi(x)\phi(y)$, $\phi(x)\psi(y)$, $\psi(x)\phi(y)$, and $\psi(x)\psi(y)$. Each step of the transform for such bases involves two frequency splits instead of one. Suppose that a N x N image is applied to the process. First each of the N rows in the image is split into a low pass half and a high pass half. The result is an N x $\frac{N}{2}$ low pass sub image and an N x $\frac{N}{2}$ high pass sub image. Next each column of the sub images is split into a low pass and a high pass half. The result is a four way partition of the image into horizontal low pass/vertical low pass, horizontal high pass/vertical low pass, horizontal low pass/vertical high pass and horizontal high pass/vertical high pass sub images. The low/low pass sub image is subdivided in the same manner in the next step. An N x N image exposed to this process is illustrated in figure 1.1. Also it will be seen that the Mallat algorithm for the discrete wavelet transform involves this procedure exactly.

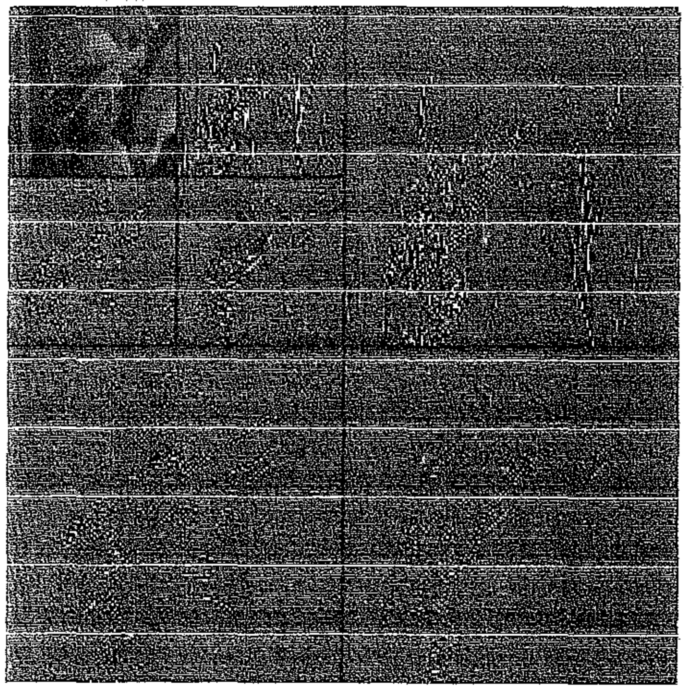


Fig. 1.1 Wavelet transform of the image "Lena"

## 1.2 DILATION

Since most of the information exits in the low pass filter output, then it is possible to transform this low pass filter output by filtering the output with another set of high pass and low pass filters. The new output of this filters is a set of data each one quarter the size of the original input. Again the low pass filter output of the 2nd dilation can be further transformed and so on.

If the number of input samples is N $= 2^D$ then a maximum of D dilations can be performed, the last dilation resulting in a single low pass value and high pass value as shown in figure 3.1. The decomposition is on a logarithmic frequency scale as opposed to the linear scale of the Fourier transform and the lowest possible frequency which can be represented by the decomposition is clearly limited by the number of samples in the block. This differs from the Fourier treatment in which the decomposition includes all frequencies down to zero due to its infinite support..

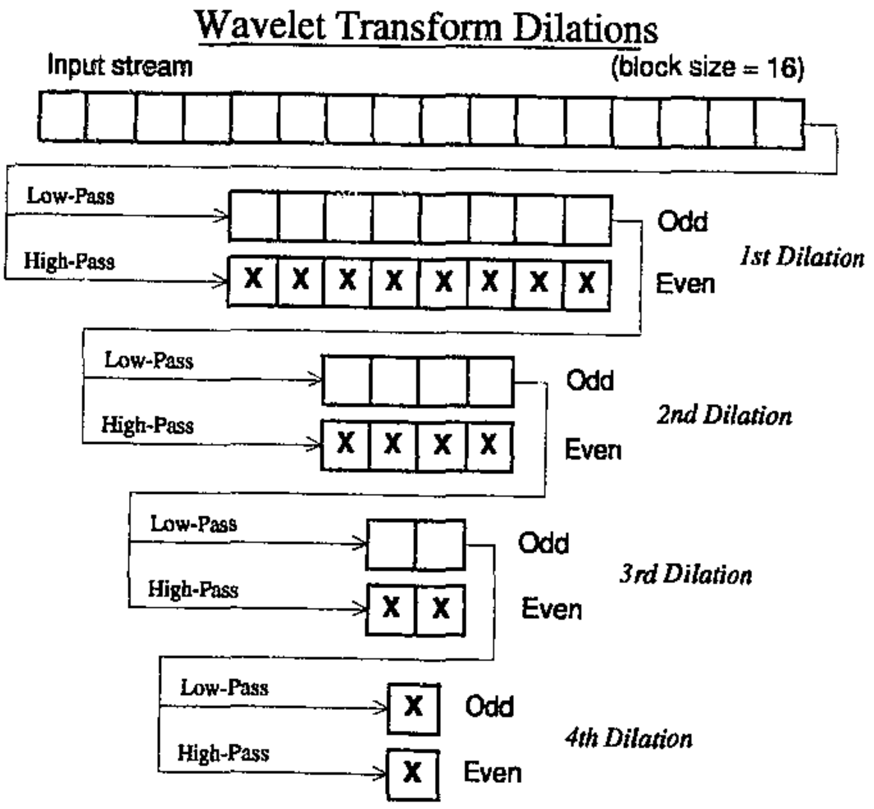


Fig. 1.2 Dilations of a sample block of data

## 1.3 IMAGE COMPRESSION

Wavelet transformed images become sparse ideally. This means that a high proportion of the matrix has zero entries or close to zero. The low pass filter output contains most of the information and the high pass filter contains flavour or nuance information, which can be discarded without affecting greatly the quality of the image. Compression is achieved when the zero elements in the sparse matrix are discarded. Ideally if the sparse matrix contains many zero entries, then the size of the matrix can be reduced considerably after removal of the zero entries. Furthermore, a non negative threshold value $\varepsilon$ can be defined for the compression system and any data whose magnitude is less than or equal to $\varepsilon$ will be reset to zero.

Lossy compression is achieved when $\varepsilon > 0$ since when the image is reconstructed it results in an approximation of the original image. The reconstructed image may be an approximation but the quality of the reconstructed image is visually acceptable. In some cases it is difficult to tell the difference visually between original and reconstructed. The second type of compression available is lossless compression, $\varepsilon = 0$. This type of compresion has the property that the reconstructed image is an exact copy of the original image without any errors. Obviously since more data can be discarded when lossy compression is used, then higher compression rates can be expected for this method of compression. Compression ratios of up to 100:1 and even greater are possible, however, the increase in compression ratio is only at the expense of degraded image quality in the reconstructed image.

## 1.4 SUBBAND TRANSFORMS

Linear transforms are the basis for many techniques used in image processing, image analysis, and image coding. Subband transforms are a subclass of linear transforms which offer useful properties for these applications. In this chapter a variety of subband decompositions will be examined and their use in image coding is illustrated.

Traditionally, coders based on linear transforms are divided into two categories: Transform coders and subband coders. This distinction is due in part to the nature of the computational methods used for the two types of representation. A subband transformer is a multi-rate digital signal processing systems. There are three elements to multi-rate systems: Analysis filters ($H_n$) , interpolators $\uparrow M$ , decimators $\downarrow M$, and Synthesis filters ($G_n$). These elements are packed in a block termed the filter bank as seen in figure 4.1.

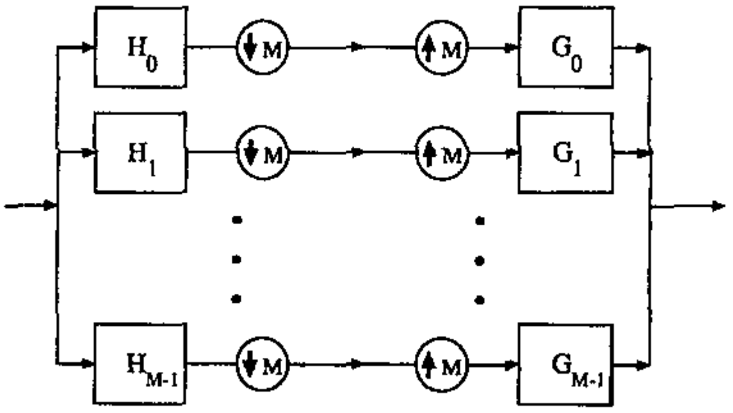


Fig.1.3 Filter bank

Transform coding techniques are usually based on orthogonal linear transforms. The Discrete Fourier Transform (DFT) is a typical transform which decomposes a signal into sinusoidal frequency components. Also, the Discrete Cosine Transform (DCT) and the Karhunen-Loeve Transform (KLT) are typical transform coding techniques. The transform is performed in most cases by taking the inner product of the finite-length signal with a set of basis functions. This produces a set of coefficients, which are then passed on to the quantization stage of the coder.

Subband transforms are generally computed by convoluting the input signal with a set of bandpass filters and decimating the results [6]. Each decimated subband signal encodes a particular portion of the frequency spectrum, corresponding to information occurring at a particular spatial scale. To reconstruct the signal, the subband are upsampled, filtered, and then combined additively.

## 1.5 TRANSFORM PROPERTIES

The criteria used in choosing a linear transformation for coding purposes should be carefully developed. There is a set of properties which are relevant when considering the problem of image coding. The properties of interest are;

- Scale and orientation
- Spatial Location
- Orthogonality

## 1.51 SCALE AND ORIENTATION

Images contain objects and features of many different sizes which may be viewed over a large range of distances. Image transformation should analyse the image simultaneously and independently at different scales. Several authors [7,8] have argued that the correct partition in terms of scale is one in which the scales are related by a fixed constant of proportionality. In the frequency domain, this corresponds to a decomposition into localised subbands with equal widths on a logarithmic scale.

For two-dimensional signals, a localised region in the frequency plane corresponds spatially to a particular scale and orientation. Orientation specificity allows the transform to extract higher order oriented structures typically found in images, such as edges and lines. Thus, it is useful to reconstruct transformations which partition the input signal into localised patches in the frequency domain.

## 1.52 SPATIAL LOCALISATION

Spatial localisation is useful where information about the location of features in the image is critical. Spatial localisation should not occur abruptly since it leads to poor localisation in the frequency domain but in most image coding systems it is advantageous to have spatial localisation.

The concept of joint localisation in the spatial and spatial-frequency domains may be contrasted with the two most common representations used for the analysis of linear systems: the sampled or discrete signal, and its Fourier transform.

The standard basis for discrete signals consists of impulses located at each sample location. The basis functions are maximally localised in space, but convey no information about scale. On the other hand, the Fourier basis set is composed of even and odd phase sinusoidal sequences, whose usefulness is primarily due to the fact that they are the eigenfunctions of the class of linear shift-invariant systems. Although they are maximally localised in the frequency domain, each one covers the entire spatial extent of the signal.

## 1.53 ORTHOGONALITY

Decorrelation is the main reason for orthogonality as a property of an image coding system. Given a signal with prescribed second order statistics, there is an orthogonal transform which will decorrelate the signal. This means that the second order correlation of the transform coefficients will be zero. Orthogonality is usually not discussed in the context of subband transform, although many such transforms are orthogonal. Hence orthogonality is not strictly necessary but in most image coders it is advantageous to have the property of orthogonality.

## 1.6 SOME TRANSFORMS

For a transform to be useful it should be well localised in the spatial and frequency domains. Furthermore, Criteria has been provided [10] for choosing a linear transformation for image coding purposes. An explicit representation of scale is widely accepted as being important for effective image representation [3]. In addition to localisation in frequency, it is advantageous for the basis function to be spatially localised. Finally, the basis should be orthogonal for proper decorrelation of the image.

In this section several transforms will be examined by considering the criteria of scale, localisation and orthogonallity. Each transform will be then evaluated for its advantageous and disadvantageous properties.

### 1.61 THE GABOR TRANSFORM

A solution to the problem of spatial localised subband decomposition is proposed by Dennis Gabor [9]. Gabor introduced a one dimensional transform in which the basis functions are sinusoids weighted by Gaussian windows. The Gabor transform can be considered to perform a localised frequency decomposition in a set of overlapping windows. The resulting basis functions are localised in both space and spatial frequency. In two dimensions, the Gabor basis functions are directional sinusoids weighted by gaussian windows. Daugman [11] has used two dimensional Gabor transforms for image compression successfully.

The problem with this transform is that the sampling functions are drastically different from the basis functions, hence the basis function is non orthogonal. In a coding application, errors introduced by quantization of the coefficients will be distributed throughout the spatial and frequency domains, even though the coefficient values are computed based on the information in localised spatial and frequency regions.

An interesting feature of the Gabor Transform is that localisation can be improved if an overcomplete Gabor basis set is used by spacing the Gaussian windows more closely than is required, or by dividing each window into more frequency bands. The use of overcomplete Gabor basis is an active area of research and several authors [12,13] have used this to compress image data.

## 1.62 THE DCT TRANSFORM

The DCT is the cornerstone of the JPEG image compression standard. In the baseline version of this standard, the image is divided into a number of 8 x 8 pixel blocks, and the block DCT is applied to each block. The resulting block DCT basis functions constitute a subband transform. The DCT has the property of packing signal energy into a small number of coefficients and is a desirable feature in most transform coders. Furthermore, the transform is orthogonal hence many of the problems of the Gabor transform are eliminated.

The problem with this transform is that although the resulting block DCT basis functions constitute a subband transforms, the subbands are not well localised. The subsampled subband images will contain severe amounts of aliasing. This aliasing is removed in the synthesis stage, however, if the transform coefficients are quantized or discarded then the aliasing is not removed and the errors appear as block edge artefacts in the reconstructed image.

Jager [14], has proposed techniques for reducing the aliasing of the block DCT by using lapping techniques. However the amount of aliasing removed is limited to equal-sized subbands. Also it is advantageous to subdivide the spectrum into equal log-width subbands in order to reduce the amount of aliasing [14].

## 1.63 THE LAPLACIAN PYRAMID

This is one of the first techniques used for octave subband decomposition as developed by Burt [17]. An octave subband transform may be constructed by cascading a two band analysis/synthesis (A/S) system in a non uniform manner as shown in figure 5.1. This system is suitable for data compression, since the multi-scale nature of the pyramid makes it particularly useful for the task of progressive transmission. In the case of a pyramid, this is easily accomplished by sending the transform coefficients in order from lowest to highest resolution.

The Laplacian pyramid suffers from similar problems to the Gabor Transform since this transform is also non orthogonal. The most serious problem with this transform is that quantization errors from highpass subbands do not remain in these subbands. Instead, they appear in the reconstructed image as broadband noise. Furthermore, the basis set is overcomplete, requiring an increase of the number of sample points over the original image. Finally, the two-dimensional basis functions are not oriented, and thus will not extract the oriented structural redundancy typically found in natural images.

Despite all the difficulties experienced by this transform, it is still considered very efficient in progressive image coding. The Laplacian pyramid has been effectively used for motion-compensated video coding, where its overcompleteness makes it robust in to motion-compensation errors [18].

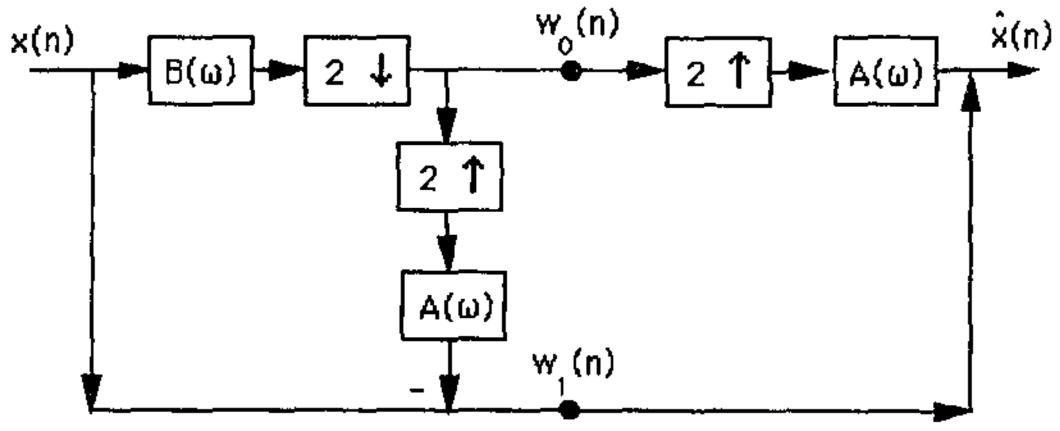


Fig. 1.4 A one level of the Laplacian pyramid. B(W) is a low pass filter and A(W) is a high pass filter

## 1.64 QUADRATURE MIRROR FILTERS

A useful two-band subband transform was developed by Croiser et. al. [19,20] and is commonly used for speech coding. This transform is based on banks of Quadrature Mirror Filters (QMF). The filters used by Croiser were a class of non-ideal FIR bandpass filters that could be used in an A/S system while still avoid aliasing in the overall system output. Aldelson et al. [21] and Mallat [3] found that these filters form an orthogonal subband transform. Mallat related QMF to the mathematical theory of wavelets and Vetterli [22] suggested the used of these filters for image coding of two dimensional images.

Transforms using QMF captures the advantages of previous mentioned transform coding techniques, while avoiding the disadvantages. It satisfies the properties of useful transform coding being: it is multi-scale and oriented, it is spatially localised, and is an orthogonal transform, and so constrains quantization errors to remain within subbands.

The QMF transform has an unfortunate aspect, the orientation decomposition is incomplete and hence the two diagonal orientations lump together in a single subband. This causes the reconstructed image to look 'blocky'. To address this problem authors like Adelson [21] have proposed using non-separable or non oriented filters [22].

## 1.65 ASYMMETRICAL QMF FILTERS

For a QMF transform, the computational complexity is directly proportional to the size of the filters employed. Separable QMF filters can sometimes ignore the issue of computational efficiency due to the steady increase in the speed of signal processing hardware. However, when considering the encoding and decoding of images using general purpose hardware, issues such as computational speed are very important.

In this situation, it is advantageous to develop asymmetric coding techniques in which simplicity is emphasised at one end at the expense of complexity most of the time. In QMF bank filters this requires designing the filters to be less orthogonal. In a A/S system the computational efficiency can be increased by using a very compact filter pair in the synthesis stage as demonstrated by Mallat and Adelson [3,21].

## 1.66 NON-SEPARABLE QMF TRANSFORMS

Most two dimensional work with QMFs employs separable or non oriented filters to achieve the transform. As discussed before, separable application of one dimensional QMFs produces a representation in which one of the subbands contains a mixture of two orientations and is a major drawback of separable QMFs. Splitting the frequencies at this subband requires very large filters. In general, the high-frequency diagonal regions of the spectra of the natural images are relatively insignificant. But if the filter bank is cascaded to form a pyramid, then the lower frequency diagonals ( where there is significant power) will also be mixed.

The non-separable QMF transform has been investigated by Adelson [21] and uses hexagonal symmetric filters to achieve non separable QMF transforms. However although improving the situation of mixed orientations, the nature of the function is blocked like the DCT, hence unlikely to offer efficient image compression.

## 1.7 MULTIRESOLUTION ANALYSES

Most linear transforms have been motivated by probabilistic considerations and assume that the image can be reasonably well-approximated by Gaussian random vectors with a particular covariance structure. The use of the wavelet transform in image coding is motivated by a rather different perspective, that of approximation theory. Wavelet transforms assume that images are locally smooth functions and can be well-modelled as piecewise polynomials. This new perspective provides some valuable insights into the coding process and has motivated some significant advances.

To illustrate the usefulness of multiresoltuion analyses and how wavelets are motivated by this perspective consider a continuous-valued square-intergrable function f(x) using a discrete set of values. A natural set of values to approximate f(x) is a set of regularly spaced, weighted local averages of f(x) such as might be obtained from the sensors in a digital camera. A simple approximation of f(x) based on local averages is a step function approximation it has the form

$$Af(x) = \sum_{n} f_n \phi(x-n)$$

where $f_n$ is the height of the step in [n,n+1] and $\phi(x) = 1$ for $x \in [0,1)$ and 0 elsewhere.

A more generalised approximation will have the form

$$Af(x)=\sum_{n}(\delta(x-n),f(x))\phi(x-n)$$

Where $\delta(x)$ is a weight function and $\phi(x)$ is an interpolating function. The restriction on $\phi(x)$ ensures that the approximation is exact when f(x) is a linear combination of the functions $\phi(x-n)$.

The resolution of the approximation of f(x) can be varied by dilating and contracting the functions $\phi(x)$ and $\delta(x)$. The approximation of Af(x) can then be formed by projecting f(x) onto the span of the functions $(\phi^j(x-2^{-j}k))_{k\in Z}$. If $V_j$ is the space spanned by this functions then the resolution j approximation $A^j f$ is simply a projection (not necessary an orthogonal one) of f(x) onto the span of the functions $\phi^j(x-2^{-j}k)$.

The approximation $A^j$ f(x) corresponds to an orthogonal projection of f(x) onto the space of step functions with step width $2^{-j}$. Figure 6.1 shows the difference between the coarse approximation $A^0$ f(x) on the left and the higher resolution approximation $A^1$ f(x) on the right.

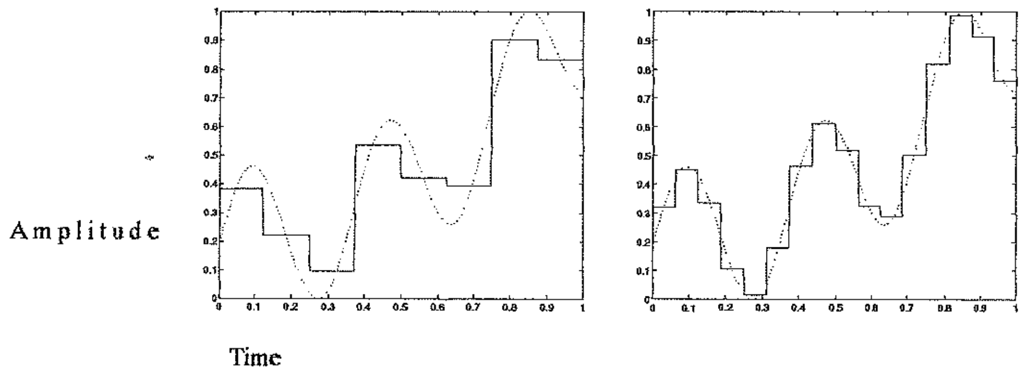


Fig. 1.5 A continuos function f(x) (plotted as a dotted line) and the $A^0$f(x) approximation. Right diagram is $A^1$ f(x) , a higher resolution approximation of f(x).

## 1.8 WAVELET TRANSFORMS VS SUBBAND DECOMPOSITION

The wavelet transform is a special case of a subband transform, as the derivation of the fast wavelet transform reveals. Wavelets involve the analysis of continuous functions whereas analysis of subband decompositions is more focused on discrete time signals. Hence, the main contribution of wavelets transform is one of perspective. The theory of wavelets has a strong spatial component whereas subbands are more focused in the frequency domain.

The subband and wavelet perspectives represent two extreme points in the analysis of the iterated filtering and down sampling process. The filters used in subband decompositions are typically designed to optimise the frequency domain behaviour of a single filtering and subsampling. Theoretically, the wavelet basis functions can be obtained by iterating the filtering and performing a down sampling procedure an infinite number of times. However in typical applications, the number of iterations is limited to the sample size. Examination of the properties of the basis functions provide considerable insight into the effects of iterated filtering.

The wavelet framework explicitly specifies an underlying continuous-valued function from which the original coefficients are derived. The use of continous-valued functions allows the use of powerful analytical tools, and it leads to a number of insights that can be used to guide the filter design process. Within the continuous-valued framework the types of functions that can be represented exactly with a limited number of wavelet coefficients are characterised. Examination of these issues have led to important new design criteria for both wavelet filters [4] and subband decompositions.

The second important difference is that wavelets involves spatial as well as frequency considerations whereas subband transform is typically more focused on the frequency domain. Also Coefficients in the wavelet transform correspond to features in the underlying function in specific, well-defined locations. The explicit use of spatial information has proven quite valuable in motivating some of the most effective wavelet coders.

## 1.9 WAVELET PROPERTIES

Whereas in subband transform the properties of scale, orthogonality and localisation are of significance, in wavelets the properties of particular interest for image coding are accuracy of approximation, the smoothness, and the support of the wavelet bases [23]. When coding natural images which tend to contain locally smooth regions, it is important that the building blocks be reasonably smooth. If the wavelets possess discontinuities or strong singularities, coefficient quantization errors will cause these discontinuities and singularities to appear in decoded images. Such artefacts are highly visually objectionable, particularly in smooth regions of images.

Procedures for estimating the smoothness of wavelet bases has been developed by Rioul [24] who has pointed that in certain conditions the smoothness of scaling functions is more important criterion than standard frequency selectivity criterion used in subband coding.

Accuracy of approximation is another important property to consider in wavelet based coders. Wavelets can construct smooth, compactly supported bases that can exactly reproduce any polynomial up to a given degree. If a continuous valued function f(x) is locally equal to a polynomial, then the portion of f(x) which is equal can be reproduced with just a few wavelet coefficients.

The degree of the polynomials that can be reproduced exactly is determined by the number of vanishing moments of the dual wavelet $\psi(x)$. This wavelet has N vanishing moments provided that $\int x^k \psi(x)dx = 0$ for $k = 0,......,N$. Compactly supported bases for $L^2$ for which $\psi(x)$ has N vanishing moments can locally reproduce polynomials of degree N-1.

The number of vanishing moments also determines the rate of convergence of the approximations $A^j f$ to the original function f as the resolution goes to infinity. It has been shown [23] that $\|f - A^j f\| \leq C2^{-jN} \|f^{(N)}\|$ where N is the number of vanishing moments of $\psi(x)$ and $f^N$ is the $N_{th}$ derivative of f.

The size of support of the wavelet basis is another important wavelet property. Suppose that the function f(x) is equal to a polynomial of degree N-1. If $\psi$ has N vanishing moments, then any basis function for which the corresponding dual function lies entirely in the region in which f(x) is a polynomial will have a zero coefficient. The smaller the support of $\psi$ is, the more zero coefficients that can be achieved. More importantly, edges produce large wavelet coefficients. The larger $\psi$ is, the more likely it is to overlap an edge hence it is important that wavelets have reasonably small support.

There is a problem with limited support in wavelets as specified by Simoncelli [21]. Wavelets with short support have strong constraints on their regularity and accuracy of approximation, but as the support is increased they can be made to have arbitrary degrees of smoothness and number of vanishing moments. This limitation on support is equivalent to keeping the analysis filters short.

Limiting filter length is also an important consideration in the subband coding literature, because long filters lead to ringing artefacts around the edges of the image. Unser [26] shows that spline wavelets are attractive for coding applications based on approximation theoretic considerations. Experiments by Rioul [24] for orthogonal bases indicate that smoothness is an important consideration for compression. Antonini at al [25] find that both vanishing moments and smoothness are important, and for the filters tested they found that smoothness appeared to be slightly more important than the number of vanishing moments.

Nonetheless, Vetterly and Herley [38] state that "the importance of regularity for signal processing applications is still an open question". The bases most commonly used in practice have between one and two continuous derivatives. Additional smoothness does not appear to yield significant improvements in coding results.

Villasenor et al [4] have systematically examined all minimum order biorthogonal filter banks with lengths $\leq$ 36 as well as the additional mentioned criteria, the oscillatory behaviour and sensitivity of the coarse-scale approximations $A^j f(x)$ to translations of the function f(x) have been examined by experiments. The best filter found in these experiments was a 7/9 tap spline variant with less dissimilar lengths from [25]. This filter is one of the most common wavelet coding filters currently used..

For biorthogonal transforms, the squared error in the transform domain is not the same as the square error in the original image. As a result, the problem of minimising image error is considerably more difficult than in the orthogonal case. A number of other filters yield performance comparable to that of the 7/9 filter of [25] provided that bit allocation with a weighted error measure is performed. One such basis is the Daslauriers-Dubuc interpolating wavelet of order 4 [27], which has the advantage of having filter taps that are dyadic rationals.

A new set of filters have been developed by Balasingham and Ramstad [28]. Their design combines classical filter design techniques from ideas from wavelet constructions. This filters yields performance significantly better than the popular 7/9 filter set from [25].

## 1.10 ENTROPY

The entropy of the signal plays an important part in image coding and compression with respect to the possible effectiveness in the storage and transfer of the signal [16]. In many cases a transformation of the signal leads to a representation of the signal which lowers entropy of the signal. Now, the influence of the used wavelet filter to the entropy of the wavelet coefficients depends on the specific distribution and is defined as follows.

$$H = - E[\mathrm{Ln}\, P_Y(y)] \tag{3}$$

With respect to the estimation of the distribution function one has to differ between the approximation and the set of the detail signals. The approximation has a strong correlation, while the coefficients of the detail signals are decorrelated extensively. For the set of detail signals one can assume a Laplace distribution of the coefficients with respect to experimental investigations [17]. In analogy to the detail signals one can fit them to the same distribution of the approximation coefficients with a predictive coding. After the predictive coding, the entropy is calculated as;

$$H_\lambda = \frac{-\lambda}{2} \int e^{-\lambda|x-x|} . (\ln\frac{\lambda}{2} - \lambda|x-x|)dx \tag{4}$$

# CHAPTER 2

# IMAGE CODERS

## 2.1 BASIC IMAGE CODERS

Most wavelet based coders are derived from the transform coder paradigm. There are three basic components that underlay current wavelet coders: a decorrelating transform, a quantization procedure, and an entropy coding procedure as seen in figure 2.1. Considerable research is being performed on all three of these component and the function that each component has in the image coder.

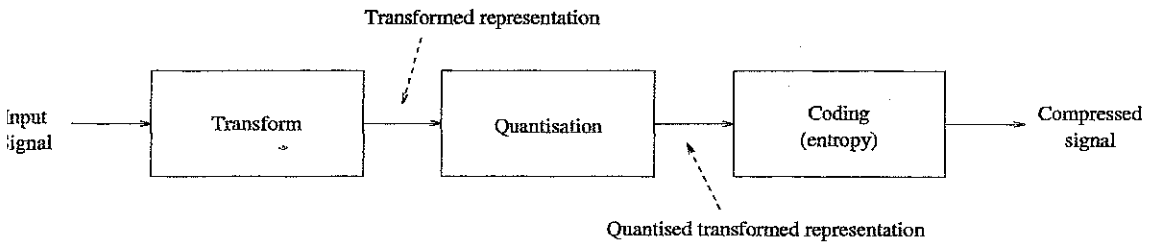


Fig. 2.1 A basic image coder

## 2.2 ENTROPY CODING AND ARITHMETIC CODING

Arithmetic Coding provides a near-optimal entropy coding for the quantized coefficient values. The coder requires an estimate of the distribution of quantized coefficients, this estimate can be approximately specified by providing parameters for a generalised Gaussian or a Laplacian density. Alternatively, the probabilities can be estimated on-line. On-line adaptive estimation has the advantage of allowing coders to exploit local changes in image statistics. Efficient adaptive estimation procedures are discussed in [29].

Because images are not jointly Gaussian random processes, the transform coefficients, although decorrelated, still contain considerable structure. The entropy coder can take advantage of some of this structure by conditioning the encodings on previously encoded values. Chen [29] presents a coder which obtains modest performance improvements using such a technique.

## 2.3 TRANSFORM CODING

Chapter 1 indicated that there are generally two types of transform coders, subband based coders, and transform based coders. Moreover subband coding has been discussed in detail in this chapter. Transform coding is used in image coders to reduce spectral redundancy by condensing the energy of an image into a small area.

Typically coders split the image into smaller blocks and a unitary transform is applied to each of the blocks resulting in the formation of a block of transform coefficients which represent the image. The transform coefficients produced by the application of a unitary transform is simply the significance of the frequency information in the image. Therefore low-frequency information will correspond to transform coefficients that are large in value, and high frequency information will correspond to coefficients that are small in value.

## 2.4 RUN LENGTH CODING

This coding method uses the statistical properties of an image to reduce redundancy but instead of generating variable length codewords, the codewords generated are of a fixed length.

## 2.5 ZEROTREE CODING

The knowledge that images of interest are formed mainly from flat area, textures, and edges allows to make advantage of the resulting cross-band structure. Zerotree coders combine the idea of cross-band correlation with the notion of coding zeros jointly to generate very powerful compression algorithms.

The first instance of the implementation of zerotrees is due to Lewis and Knowles [30]. In their algorithm, the image is represented by a tree structured data . This data structure is implied by a dyadic discrete wavelet transform in two dimensions. The zerotree quantization model used by Lewis and Knowles was arrived at by observing that often when a wavelet coefficients are small, its children on the wavelet tree are also small. This phenomenon happens because significant coefficients arise from edges and textures, which are local. It is not difficult to see that this is a form of conditioning. Lewis and Knowles took this conditioning to the limit, and assumed that insignificant parent nodes always imply insignificant child nodes. A tree or sub-tree that contains only significant coefficients is known as zerotree.

The most significant contribution of the work by Lewis and Knowles was to realise that wavelet domain data provide an excellent context for run-length coding: not only are a large run lengths of zeros generated, but also there is no need to transmit the length of zero runs, because they are assumed to automatically terminate at the leaf nodes of the tree. Much the same as in JPEG, this is a form of joint vector/scalar quantization. Each individual coefficient is quantized separately, but the symbols corresponding to small coefficients in fact are representing a vector consisting of that element and the zero run that follows it to the bottom of the tree.

Lewis and Knowles assumed that small parents always have small descendants. However, this assumption causes large distortions in image reconstruction if the small parents don't have small children. This has led many authors like Shapiro[2] to develop a much more powerful algorithm called, Embedded Zerotree Wavelet Algorithm, which overcomes many of the difficulties of the Lewis and Knowles coder.

## 2.6 THE EMBEDDED ZEROTREE WAVELET ALGORITHM

The Lewis and Knowles algorithm, while capturing the basic ideas inherent in many of the later coders, was incomplete. It had all the intuition that lies at the heart of more advanced zerotree coders, but did not efficiently specify significance maps, which is crucial to the performance of wavelet coders.

The first algorithm to employ both the non zero data and significance map was produced by Shapiro [2] and was to revolutionise the way more advanced coders operate. The bits needed to specify a significance map can easily dominate the coder output, especially at lower bitrates. However, there is a great deal of redundancy in a general significance map for visual data. Therefore the bitrates for its representation can be kept in check by conditioning the map values at each node of the tree on the corresponding value at the parent node. Whenever an insignificant parent is observed, it is highly likely that the descendants are also insignificant. Therefore, most of the time a "zerotree significance" map symbol is generated. But because p, the probability of this event, is close to 1, its information content, -p log p, is very small. So most of the time, a very small amount of information is transmuted and this keeps the average bitrate needed for the significance map relatively small.

Once in a while, one or more of the children of an insignificant node will be significant. In that case, a symbol for "isolated zero" is transmitted. The likelihood of this event is lower, and thus the bitrate for conveying this information is higher. But it is essential to pay this price to avoid losing significant information down the tree and therefore generating large distortions.

In summary, Shapiro's algorithm uses three symbols for significance maps: zerotree, isolated zero, or significant value. By using this structure, and by conditionally entropy coding these symbols, the coder achieves very good rate distortion performance. In addition, Shapiro's coder also generates an embedded code. Coders that generate embedded codes are said to have the progressive transmission or successive refinement property. Successive refinement consists of first approximating the image with a few bits of data, and then improving the approximation as more and more information is supplied. An embedded code has the property that for two given rates R1>R2, the rate R2 code is a prefix to the rate R1 code.

The EZW encoder can easily achieve a precise bitrate because it continues to output bits when it reaches the desire rate. Furthermore, it can cease decoding at any point, generating an image that is the best representation possible with the decoded number of bits. This is of practical interest for broadcast applications where multiple decoders with varying computational, display and bandwidth capabilities attempt to receive the same bitstream. With an embedded code, each receiver can decode the passing bitstream according to its particular needs and capabilities. Also the EZW coding system is useful in indexing and browsing applications, where only a rough approximation is sufficient for deciding whether the image needs to be decoded or received in full.

Shapiro generated an embedded code by using a bit-slice approach. In the bit-slice approach  wavelet coefficients are indexed into a one dimensional array and ordered according to their significance. This places the lower frequency contents of the image before the high frequency bands. When wavelet coefficients are encoded using their order of importance, it is sometimes referred to as a raster scan order.

The bit-slice code is generated by scanning the one dimensional array and comparing each coefficient to a threshold T. This initial scan provides the decoder with sufficient information to recover the most significant bit slice. In the next pass, information about each coefficient is refined to a resolution of T/2, and the pass generates another bit slice of information. This process is repeated until there are no more slices to code..

The upper bit slice contains a great number of zeros because coefficients at this level are either zero or bellow the threshold level T. The role of zerotree coding is to avoid transmitting all these zeros. Once a zerotree symbol is transmitted, it is known that all the descendants will also be zero coefficients, so no information is transmitted for them. In effect, zerotrees are a clever form of run-length coding, where the coefficients are ordered in a way to generate longer run lengths as well as making the runs self terminating so the length of the runs need not be transmitted.

The zerotree symbols with high probability and small code length can be transmitted again and again for a given coefficient until it rises above the sinking threshold, at which point it will be tagged as a significant coefficient. After this point, no more zerotree information will be transmitted for this coefficient.

Shapiro used a clever method of encoding the sign of the wavelet coefficient with the significance information. Further details of the priority of wavelet coefficients, the bit-slice coding, and adaptive arithmetic coding of quantized values, i.e. entropy coding can be read in Harder's thesis [21]

# CHAPTER 3

# SEARCH STRATEGIES AND EZW COMPONENTS

## 3.1 TREE SEARCH STRATEGIES

The EZW algorithm establishes an ancestor-decendant relationship between the wavelet coefficients in the image subbands . Figure 4.1 shows the image subbands obtained after the 2D-DWT and an ancestor-decendant tree hierarchy between the wavelet coefficients in the image subbands has been applied to an image. The scanning of the coefficients is done in a particular manner so that no child node is scanned before its parent is scanned. In essence, all the coefficients in a subband are encoded before encoding the coefficients in another subband. The order of encoding begins from the lowest frequency subband and ends at the highest frequency subband as indicated.

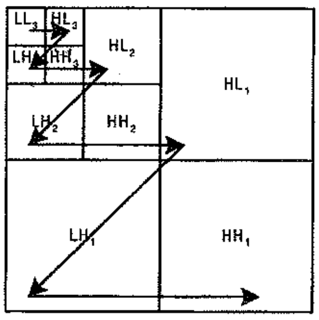


Fig. 3.1

Raster Scanning of coefficients

## 3.2 DEPTH FIRST SEARCH AND BREADTH FIRST SEARCH

The original EZW algorithm uses a raster scan to encode the wavelet coefficients as seen in figure 4.1. Two other tree searching strategies to encode the wavelet coefficients are possible: the Depth First Search (DFS), and the Breadth First Search (BFS). These strategies provide alternative ways to encode the wavelet coefficients.

The BFS coding strategy operates very similar to the raster scan strategy. Coefficients are also scanned at the same level from all tree hierarchies before the coefficients from another level are encoded. The main difference is the way or the order in which the coefficients are scanned in the same level of the tree hierarchy.

The DFS coding strategy encodes all the coefficients in a tree hierarchy before encoding another tree hierarchy. This simplifies and diversifies the implementation of the EZW algorithm [32]. However, this simplification comes at the cost of a decrease in coding efficiency but provides a dataflow oriented approach which is highly suitable for parallel architectures. The main reason for the parallel suitability is that the DFS performs a natural partitioning of the tree hierarchies into independent tree hierarchies which can then be processed in parallel by individual EZW processors as seen in figure 3.2.

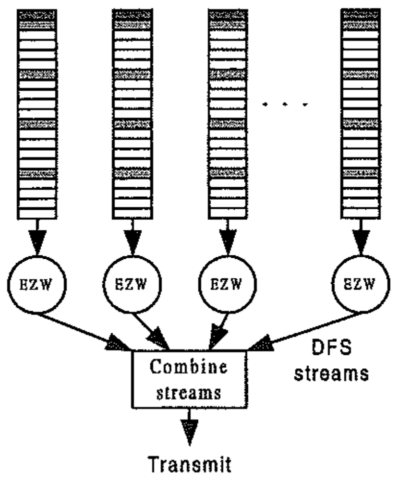


Fig. 3.2

Parallel Bitstream Processing by individual EZW Processors

## 3.3 EZW COMPONENTS

The EZW algorithm is a powerful image coding technique. It has been shown [2] that the algorithm consists of four fundamental operations;

1. A discrete Wavelet Transform or hierarchical subband decomposition.
2. Prediction of the absence of significant information across scales by exploiting the self-similarity inherent in images.
3. Entropy-coded successive approximation quantization.
4. Adaptive arithmetic coding.

Furthermore, prediction of significant information and entropy coding are commonly combined in a single structure termed Quantization zerotree as seen in figure 3.3.

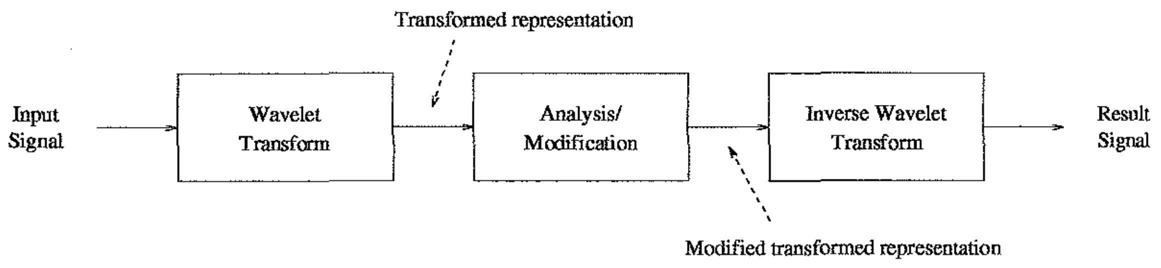

Transformed representation
Input Signal
Wavelet Transform
Analysis/ Modification
Inverse Wavelet Transform
Result Signal
Modified transformed representation

Fig. 3.3

EZW coding system block structures

## 3.31 DISCRETE WAVELET TRANSFORM

The Discrete Wavelet Transform (DWT) is one of the most useful and efficient tools used to analyse digital signals in various signal processing areas. One major concern, in signal and image processing as well as the communication communities is the effective implementation of the wavelet transform and advanced tools for designing wavelet systems. When there are limitations in processing time and/or system size, implementation of the DWT becomes an engineering issue. The purpose of this thesis is hence to provide an engineering solution to the implementation of the discrete wavelet transform. Furthermore, it will be seen how the proposed solution fits well in the proposed DFS EZW coding system seen in figure 3.3.

The Discrete Wavelet transfer represents an arbitrary square integral function as superposition of a family of basis functions called wavelets. A family of wavelet basis functions can be generated by translating and dilating the mother wavelet corresponding to the family. The DWT coefficients can be obtained by taking the inner product between the input signal and the wavelet functions. Since the basis functions are translated and dilated versions of each other, a simpler algorithm, known as Mallat's tree algorithm or pyramid algorithm, has been proposed [3].

## 3.32 MALLAT'S TREE ALGORITHM

In this algorithm, the DWT coefficients of one stage can be calculated from the DWT coefficients of the previous stage, which can be expressed as follows:

$$W_l(n,j) = \sum_m W_l(m,j-1)h(m-2n) \quad (5a)$$
$$W_h(n.j) = \sum_m W_h(m,j-1)g(m-2n) \quad (5b)$$

Where:

. $W_l(n,j)$ and $W_h(j)$ are the n-th scaling coefficient at the j-th stage.

. $h(n)$ are the Low Pass Filter (LPF) dilating coefficients.

. $g(n)$ are the High Pass Filter (HPF) dilating coefficients.

For computing the DWT coefficients of the discrete-time data, it is assumed that the input represents the DWT coefficients of a high resolution stage. Equation 5 can then be used for obtaining DWT coefficients of subsequent stages. In practice, this decomposition is performed only for a few stages. Hence, DWT extracts information from the signal at different scales. The first level of the wavelet decomposition extracts the details of the signal ( high frequency components) while the second and all subsequent decompositions extract progressively coarser information, lower frequency components, as shown in figure 3.4

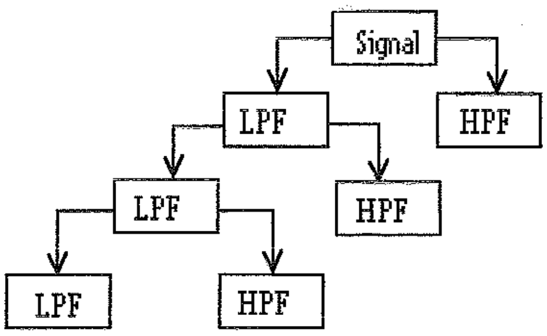


Figure 3.4

Mallat Algorithm For DWT

## 3.4 TWO DIMENSIONAL DISCRETE WAVELET TRANSFORM (2D-DWT)

The 2D-DWT can be calculated by several different methods. Most commonly, it is calculated by using a separable approach [3]. First, the One Dimensional Discrete Wavelet Transform (1D-DWT) is performed on each row of the image proceeded by a matrix transposition operation [33]. The transposition memory works on rows of the image to invert the image, hence rows become columns and vice versa. Finally, a one dimensional DWT is performed on the transposed data to achieve two dimensional data. Hence, a 2D-DWT can be implemented by inserting a matrix transposer between two 1D-DWT modules.

In order to reconstruct the original data, the DWT coefficients are upsampled and passed through another set of lowpass and highpass filters which is expressed as:

$$W_l(n,j) = \sum_k W_l(k,j+1)h_o(n-2k)+\sum_l W_h(l,j+1)g_o(n-2l) \qquad (6)$$

where $h_o(n)$ and $g_o(n)$ are the lowpass and highpass synthesis filter corresponding to the mother wavelet. It is observed in equation 6 that the j-th level DWT coefficients can be obtained from the (j+1)-th level coefficients.

## 3.5 QUANTIZATION ZEROTREE

The quantization step in the EZW algorithm involves transforming the coefficients array into a quantization zerotree structure. From this zerotree representation, a compressed data stream called a significance map representing the image is obtained using Successive-Approximation entropy-coded Quantisation (SAQ). The output of this block consists of significance maps (MAP) and Successively Approximated values of significant coefficients SAQ. SAQ applies a sequence of threshold successively to determine the significance of the coefficients to obtain a MAP.

## 3.6 ARITHMETIC CODING

The underlying reason for the choice of successive approximation to encode the significance maps for the EZW algorithm is considered directly from the goal of producing an embedded code comparable to the binary-representation of approximating a real number [2]. In achieving a reasonable coding efficiency with Huffman coding, the sequence that is generated by the source is generally divided up into blocks. Each of the blocks then get assigned a variable-length codeword. When decoded, the received codeword is parsed into variable-length blocks which correspond to the individual codewords. This causes a one-to-one correspondence between the codeword blocks and the source sequence blocks. Arithmetic coding on the other hand generates non-block codes. The entire sequence of source symbols is assigned a single arithmetic codeword.

An interval of real numbers between the values of 0 to 1 is defined by the codeword itself. As the number of symbols increases, the interval used to represent it becomes smaller and the number of bits required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence.

# CHAPTER 4

## 4.1 TWO DIMENSIONAL DWT ARCHITECTURES

Multiresolution analysis is an essential part of the EZW coding system. The output of this block consists of coefficients which are mapped in some way by the EZW quantizer. In developing a suitable 2D-DWT architecture, it is important to consider how this architecture fits with the EZW coding system. Moreover, it has been shown that the DFS searching strategy for the coefficients provides a dataflow oriented property which is highly suitable for parallel EZW processors [32].

The purpose of this thesis is to provide an efficient design for implementing the wavelet transform. Furthermore, to fit with the DFS EZW system the proposed architecture operates in a parallel manner which increases the speed of the architecture. Ideally, the factor of speed up achieved by parallelism should not cost more than a similar factor in area. In order to achieve an optimum design several current architectures are discussed in detail before the proposed architecture is considered.

The design employed in this thesis is aimed at special purpose custom single chip design. Hence, the issue of chip area and processing time is very important. Many other important design issues will also be considered .

## 4.2 COMPUTATIONAL COMPLEXITY OF THE DWT

As observed in equation 5, the complexity of each stage of wavelet decomposition is linear in the number of input samples where the constant factor depends on the length of the filter. For a dyadic wavelet decomposition, the number of input samples decreases by 50 % at subsequent stages of decomposition [35]. For a wavelet of order L, with number of decomposition stages j, the computational complexity for an one dimensional N elements sequence is.

$$C_{dyadic} = (N + \frac{N}{2} + \frac{N}{4} + \ldots\ldots + \frac{N}{2^{J-1}}) \text{x } 2L \text{ FLOPS}$$

$$= 4(1\text{-}2^{-J})NL \text{ FLOP} \qquad (7)$$

Where FLOP corresponds floating point operations and usually refers to multiplications and additions. It can be pointed out that the complexity can be further be reduced using sophisticated algorithms, such as First Running FIR Filtering (FFT) [34]. However, these algorithms need complex control circuitry for hardware implementation.

In many applications, a regular tree, instead of a dyadic tree might be more appropriate. The computational complexity at each stage of a regular tree is 2NL FLOP. Hence, the total complexity for a j level decomposition is:

$$C_{regular} = 2\ JNL \text{ FLOP} \qquad (8)$$

The complexity of an irregular tree, or a wavelet packet algorithm is upper bounded by $C_{regular}$.

## 4.3 DATA DEPENDENCIES WITHIN DWT

The Mallat tree decomposition is one of the most common methods used to implement the DWT since it is computationally efficient and can be implemented easily. The wavelet decomposition of a 1D signal for three stages as shown in figure 4.1 hints that there is a dependency of data as data flows from lower levels to higher levels. In order to implement a design, it is important to consider the data dependencies involved in the tree decomposition..

The low pass, H(Z), and high pass, L(Z), transfer functions for an n-th order filter can be expressed according to:

$$H(Z)= g(0) + g(1)z^{-1} + g(2)z^{-2} + \dots\dots\dots g(n)z^{-n} \qquad (9)$$

$$L(Z)= g(0) + g(1)z^{-1} + g(2)z^{-2} + \dots\dots\dots g(n)z^{-n} \qquad (10)$$

It is not hard to define the data dependencies by considering equations 9 and 10. Letting a, b, c, d, e, f, g represent the intermediate and final DWT coefficients (assuming a six tap filter) as seen in figure 4.11 below we have:

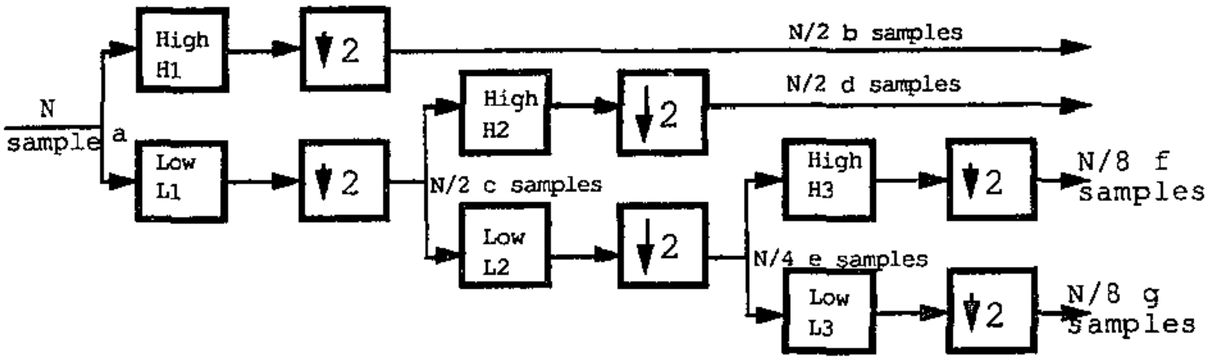


Figure 1

Data dependencies in filter bank

1st OCTAVE

$$b(0) = g(0)a(0) + g(1)a(-1) + g(2)a(-2) + \dots\dots g(5)a(-5) \qquad (11a)$$

$$b(2) = g(0)a(2) + g(1)a(1) + g(2)a(0) + \dots\dots g(5)a(-3) \qquad (11b)$$

$$b(4) = g(0)a(4) + g(1)a(3) + g(2)a(2) + \dots\dots g(5)a(-1) \qquad (11c)$$

$$b(6) = g(0)a(6) + g(1)a(5) + g(2)a(4) + \dots\dots g(5)a(1) \qquad (11d)$$

$$c(0) = h(0)a(0) + h(1)a(-1) + h(2)a(-2) + \dots\dots h(5)a(-5) \qquad (11e)$$

$$c(2) = h(0)a(2) + h(1)a(1) + h(2)a(0) + \dots\dots h(5)a(-3) \qquad (11f)$$

$$c(4) = h(0)a(4) + h(1)a(3) + h(2)a(2) + \dots\dots h(5)a(-1) \qquad (11g)$$

$$c(6) = h(0)a(6) + h(1)a(5) + h(2)a(4) + \dots\dots h(5)a(1) \qquad (11h)$$

2nd OCTAVE

$$d(0) = g(0)c(0) + g(1)c(-2) + g(2)c(-4) + \dots\dots g(5)c(-10) \qquad (11e)$$

$$d(4) = g(0)c(4) + g(1)c(2) + g(2)c(0) + \dots\dots g(5)c(-6) \qquad (11f)$$

$$e(0) = h(0)c(0) + h(1)c(-2) + h(2)c(-4) + \dots\dots h(5)c(-10) \qquad (11g)$$

$$e(4) = h(0)c(4) + h(1)c(2) + h(2)c(0) + \dots\dots h(5)c(-6) \qquad (11h)$$

3rd OCTAVE

$$f(0) = g(0)e(0) + g(1)e(-4) + g(2)e(-8) + \dots\dots g(5)e(-20) \qquad (11i)$$

$$g(0) = h(0)e(0) + h(1)e(-4) + h(2)e(-8) + \dots\dots h(5)e(-20) \qquad (11j)$$

As seen in the above equations, several intermediate results (c,e) are first computed, and then these are used to calculate multiple output samples. The intermediate results must be available for further processing at a specific time instant, implying a memory requirement in the proposed architecture design.

## 4.4 FINITE PRECISION EFFECT

The accuracy of the DWT coefficients is determined by both the precision of the input data, and the wavelet filter coefficients. Reconstructed Image quality measurements results depend proportionally on finite precision effects. Moreover data and coefficient precision are important parameters in the design of DWT architectures. High accuracy is required to achieve suitable image quality, but hardware constraints only permit a certain level of accuracy and thesis paper intends to investigate how accuracy of filter and data coefficients affect image quality in order to produce an optimal design.

In terms of hardware requirements, the DWT coefficients are recursively calculated using equation 5, where W represents wavelet coefficients of a certain stage and h, g represent the corresponding filter coefficients. If The precision of W and h are assumed to be j and m bits respectively, then to execute equation 5, a j x m bits multiplier and accumulator is required.

A useful measure of accuracy of DWT coefficients is the Signal to Noise Ratio (SNR). Here, the signal is the floating point DWT coefficient and noise is the difference between the floating point and finite precision coefficients. In a particular design the performance variation for 1D signal with respect to the precision of filter coefficients with fixed 12 bits DWT coefficients and data was observed to be 50-70 dB SNR [15]. Furthermore, it was also observed that when 2D data was involved a decrease in the SNR results.

## 4.6 IMAGE QUALITY

Signal To Noise Ration (SNR) was briefly described as a useful measure of the accuracy of DWT coefficients. In practice this is not the only available method to measure image quality. Other image quality measurements are available to access quality of reconstructed image. These measurements provide powerful tools for accessing performance of DWT architectures.

### 4.61 SUBJECTIVE AND OBJECTIVE IMAGE QUALITY

Objective measuring employs formulas that represent certain information about the image so that a comparison can be made. In other words these formulas relate the original image to the reconstructed image in some way. In particular, the Root-Mean-Square-Error (RMSE), and the Peak Signal-to-Noise Ratio (PSNR) are very common formula applied to reconstructed images. Given an image represented by f and the reconstructed image represented by g, the RMSE is given by:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i-1}^{n} \sum_{j-1}^{n} [f(i,j)-g(i,j)]^2}{N^2}} \qquad (12)$$

This value is measured in dB and is the standard deviation of the error of the reconstructed image from the original image. Also, PSNR is a parameter derived from the RMSE as follows:

$$\mathrm{PSNR} = 20 \log_{10} \left(\frac{I}{RMSE}\right) \qquad (13)$$

Where I is the maximum image intensity, i.e. for a 8 bit pixel it would be $2^8$-1 = 255. Also a further parameter is termed Mean Square Error (MSE) and is simply found by squaring the RMSE value.

Typical PSNR values have been found [31] for different compression techniques. These values are typical of good performing architectures. It is desirable that the proposed architecture produces similar results to the ones seen in table 4.1 bellow.

| Building Block | Compression Technique | Bit Rate Bits/Pixel | PSNR(dB) |
|---|---|---|---|
| Predictive coding | 2D Differential | 1 | 27.74 |
| Vector Quantization | Level 0 mean residual tree structured VQ | 0.5 | 26.9 |
| Transform Coding | JPEG DCT | 0.5 | 34.69 |
| Subband Coding | Subband Coding with VQ | 0.56 | 32.71 |
| Subband Coding | Subband Coding with EZW | 0.50 | 36.28 |

Table 4.1

PSNR for different compression techniques

Generally, the higher the PSNR the better the results obtained. Table 4.1 suggests that the EZW coding technique is one of the best techniques to achieve a high PSNR. A PSNR of 30 dB or higher is considered sufficient for most good quality image coding systems.

Subjective image quality measurements use the most crucial test, the human eye. The eye is very sensitive to poorly reconstructed images, many artefacts can be distinguished very quickly, hence it can be a valid tool to measure image quality.

# CHAPTER 5
# DWT ARCHITECTURES

## 5.0 ARCHITECTURE CONSIDERATIONS

The aim of this thesis is the implementation of discrete wavelet transforms through an efficient architecture. Several designs are researched, these share a common criteria or have the same objectives; increasing throughput, decreasing area and attempt to make better use of computational components within the system.

Five important arquitectures will be discussed before a proposed architecture is presented. The existing architectures have unique features which makes them suitable for research. It is hoped to extract these beneficial features from the existing architectures in order to produce the recommended model.

## 5.1 SYSTOLIC ARCHITECTURE

Kung defines a systolic system as a network of processors which rhythmically compute and pass data through the system [35]. In other words, the data within the system is 'pumped' through the processors, which perform simple, local computations on the data. Furthermore, although the operations being performed may not be equal in all processors for a particular time-step, the data-movement and computations throughout the complete system are in synchronisation. The connections in the network are simple and local, and there is at least a single time-step required to move data from one processor to the next. The major benefits of this architecture as reported in [35] are:

. Simple and regular connectivity

. Concurrency.

. Local communication operations

. Balanced I/O and computation

Furthermore, Systolic designs are typically highly pipelined, providing a high throughput for image coding applications. The architecture shown in figure 5.1 implements the 1D DWT and requires a transposition memory to fully implement the 2D DWT. In other words, to implement the 2D DWT transform, two modules (one for row transform and another for column transform) are used along a transposition memory [33]. Much scheduling is required if the 2D DWT is to be achieved by this architecture, hence most systolic designs are employed in one dimensional architectures.

Systolic designs require $O(n^2)$ area (dominated by the storage bank) and $O(n^2)$ time where n is the number of pixels in the image. Vishwanath [35] has produced some of the most interesting implementations of the systolic based architectures including the 2D DWT Direct Implementation Architecture, and also the popular Systollic-Parallel Architecture.

The effects of block filtering are also discussed by Vishwanath. Blocking allows the selection of the required data-word storage ( by adjusting the block size), but requires some considerable attention to the edges of the blocks to ensure that blocking artifacts are not introduced. The hierarchical nature of the 2D DWT complicates the edges considerations, and Vishwanah also noted that the required blocked input to the system is not convenient or typical of an image processing system.

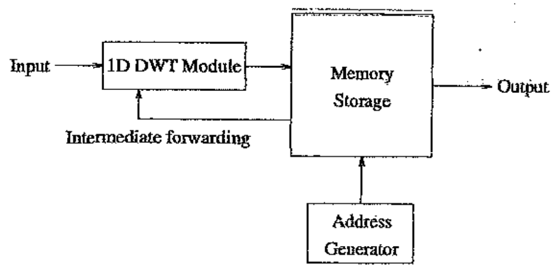


Fig. 5.1

Systolic type architecture

## 5.2 LATTICE FILTER TYPE ARCHITECTURE

A very simplistic architecture based on lattice filters, as shown in figure 5.2, is employed in [36]. This architecture is based on properties that the four ( in this case) coefficients Daubechies wavelet coefficients yields. The parallel pipeline is typically used in lattice type filters in order to get a high throughput. Furthermore, the pipelines are balanced, which means the parallel pipelines have almost the same number of process stages thus the pipeline latency and the registers in the circuits can be saved.

A definite advantage of this type of architecture is that multiplying operations, typical of other systems, are replaced by shifting and adding operations and no shift register are used. Shifting in this case is realised by the direct connections of the filter due to the constant shifting bits.

The architecture has a problem with overflow which in order to avoid, it is required that suitable word length of the address are further reduced. A further disadvantage is that the architecture is highly non modular and a complete new design is usually required to achieve higher filter lengths.

The architecture shown in figure 5.2, and reported in [36] has a high throughput of 2 outputs per clock and 5 clocks pipeline latency. The synthesised architecture contains 5058 gates and can reach 110 M pixels/s when LSI-10k CMOS technology is used.

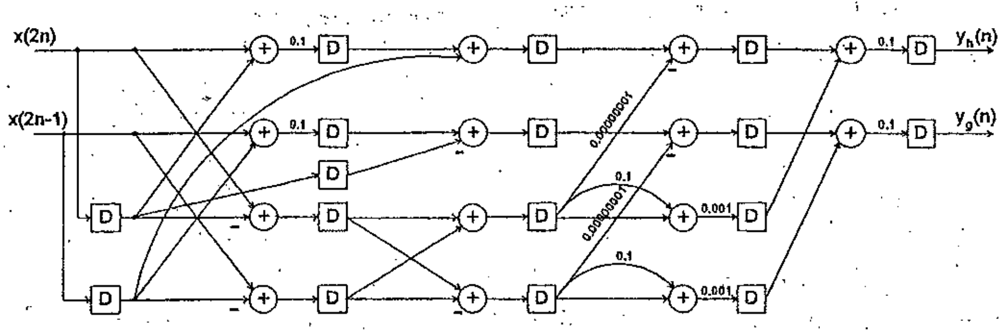


Fig. 5.2

Lattice type architecture

## 5.3 PARALLEL ARCHITECTURES

While the systolic architecture is very simple and follows a dataflow oriented approach, it lacks the high speed benefits which can be expected when data is processed in parallel. Bae and Prasanna [37] have described a two dimensional architecture aimed at parallel I/O and is highly suitable for high frame rates. For a block size of $b << n$ where n is the number of pixels in the image, this design uses b filters in parallel as a row filter, and a similar ( but slightly larger) sized column filter. Double buffering techniques are used to allow new results of the row filtering to be stored while the previous results are being column-filtered in parallel.

A typical parallel architecture is shown in figure 5.3. This model consists of three parallel filters and a storage unit. The parallel filter structure is similar to other parallel architectures presented by other authors. First, the horizontal filter HF is used to generate a highpass filtering output (H) at a positive clock cycle. During the negative clock edge, the HF filter generates the lowpass filtering output  These two resultant filtering outputs will then be stored in the corresponding lowpass (LR) and High Pass (HR) register banks. Then, the two horizontal filtering outputs (H and L) stored in the above register banks can be further decomposed by the vertical filters VF1 and VF2 generate the four vertical filtering outputs to achieve the dimensional transform.

The operation of the vertical filter is exactly the same as the horizontal filter, the highpass and lowpass filtering output of data from the horizontal filter is processed at the respective clock cycle. Finally, all the resolution levels of the DWT's can be iteratively generated by the filters HF, VF1 and VF2, where a current level of filtering data is computed in the exact  idle cycles of the previous level.

The storage unit is typically composed of several registers banks. Each register bank is used to temporarily store the filtering data that have been output from the horizontal filter. Input for the vertical filtering computations is therefore coefficients from the horizontal filter stage.

In summary 2D-DWT Parallel architectures work very similar to systolic type architectures, many of the systollic model benefits apply to parallel models. Again this means filtering data and then recursively feeding the low part component to the filter network. In the parallel filter case, the most valuable feature is that data can be processed in parallel, hence increasing the throughput of the system. But the increase in speed is only at the cost of extra hardware which may be important when considering single chip designs.

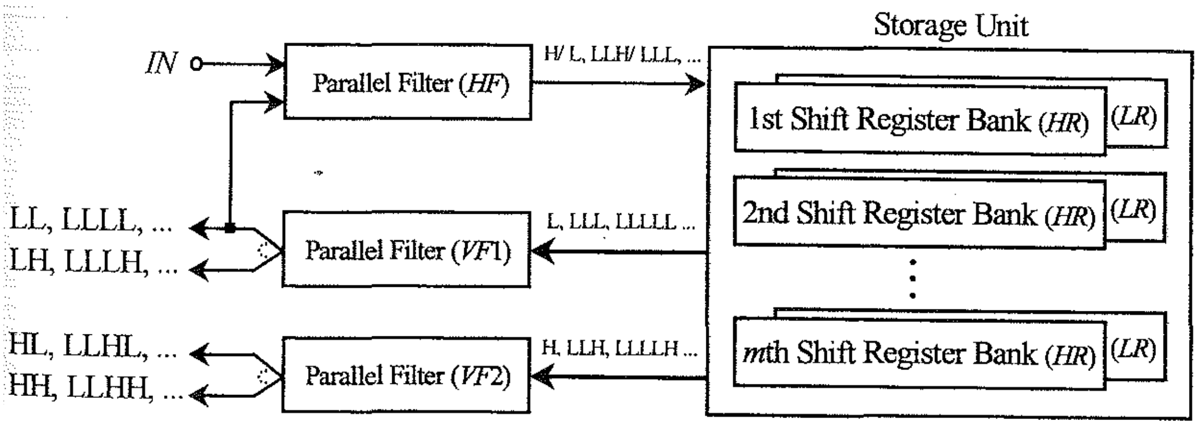


Fig. 5.3

Typical parallel filter architecture.

Typical parallel architectures need around $N^2$ clock cycles to compute all the resolution levels of a DWT where N is the number of pixels in the image. Furthermore, low cost can be expected in terms of components required to implement the architecture; a programmable parallel filter, a storage unit, and a control unit, being the only few components required. However, as seen in figure 5.3, this architecture has a large number of adders, multipliers, and registers. It has been shown that the number of elements in the architecture is highly dependent on the size of the filters employed [37]. For large filter lengths this architecture may not be suitable for single chip designs.

## 5.4 2D-DWT PROCESSOR TYPE ARCHITECTURES

Processor type architectures are becoming more predominant in the field of image coding [38]. Previously, processor designed architectures suffered from the very high access times caused by memory components. Also, expensive high speed processors were required in order to achieve the computational power required to perform the 2D DWT. More important, the cost of these systems was impractical for any worthwhile commercial program. However, as technology has progressed, the cost and speed of processors have significantly changed. Today, typical processor architectures can achieve significant speed at a very low cost. A processor designed by Chen Xuyun and others [38] can reach a computation speed of 4 M pixels/s with 7140 gates fitted on a single Field Programmable Field Array (FPGA). LSI_10K CMOS technology was used in this design, higher packing densities are assured as technology progresses, ensuring higher future computational speeds..

Processor designs are very simple, as seem in figure 5.4, a frame memory to store the image data and a 2D-DWT processor are the only elements. The processor generates, amongst other things, control signals, and row/column addresses to process the image data stored in frame memory. The 2D-DWT processor is typically divided into two parts, as seen in figure 5.4. The first part is a 2D controller, and the second part is a 1D-DWT processor.

Processor type designs rely heavily on data shifting from frame memory to the right processor for computation of the DWT coefficients. Correct scheduling of data from memory to the processor is important to produce correct results. The control unit which takes care of the correct scheduling and effectively controls of the data path of the system, consists of a 1D controller and a 2D controller. The 1D controller controls the data path unit to process 1D-DWT coefficients, the 2D controller controls the 1D-DWT processor to process the 2D DWT coefficients accordingly.

In order to improve the computational speed of processor designs architectures, many methods such as pipelining, and other clever techniques are employed. Typically, the timing relation between read/write operation and data calculation can be arranged delicately. This means that while the data is being calculated in the Data Path Unit, the controller can be reading the data which will be calculated in the next time and write the result data which has been obtained from the last calculation. Obviously this 'multitasking' of operations requires very careful consideration and planning to the overall control and data path unit of the system.

The processor type design architecture is ideal in producing systems that require a Digital Processing (DP) type architecture. This means that the architecture can easily be programmed to account for different situations, for instance the way data is taken through the data path, the DWT coefficients calculations, and even how the overall operation is performed can easily be accomplished. This gives indication that this architecture is highly suitable for researching 2D-DWT architectures. Because the architecture is highly portable, the best DWT filter coefficients could be easily evaluated. Also, due to its Digital Processing nature, it is likely that it can be easily adapted for producing coefficients in a given way, such as required by the DFS-EZW algorithm.

The best example of a commercial implementation for the wavelet transform on a processor type architecture is Aware's Wavelet transform processor (WTP) [38]. This is a single chip that is capable of 2, 4 or 6 tap wavelet transforms and is cascable for larger filter widths. However, Control is required for any further stages, i.e. handshaking between the blocks. The processing is performed by a 16 bit, 4-stage filtering pipeline, and the control of the chip is largely software based. Daubechies coefficients are built on to the WTP, or the user can pre-load their own selection of analysis or synthesis wavelet coefficients. On the other hand the designer of this architecture make no attempt to mention the high latency involved when the WTP is used to process 2D-DWT coefficients.

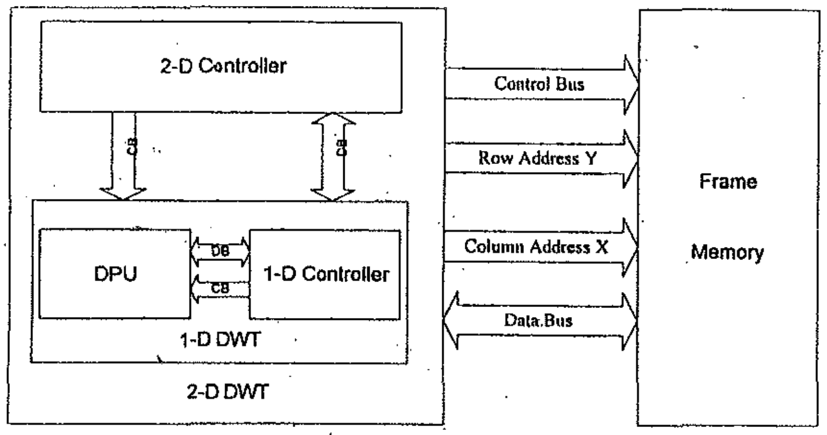


Fig. 5.4

Processor designed architecture

## 5.5 RECURSIVE ARCHITECTURE

A very popular architecture has been described as a Recursive type architecture [39]. Some of its most useful features are cost effectiveness, optimised data-bus utilisation, scheduling control overhead reduction, and storage size reduction.

The Architecture behaves closely with the way Mallat's pyramid algorithm performs the 2D-DWT. Furthermore, the operation is similar to the systolic filter architecture in which data 'flows' through the system as it is transformed. Little memory is used to hold the DWT coefficients and the only Latency experienced by such a system is caused by the transposition memory as shown in figure 5.5.

The architecture shown here uses the Daubechies four tap DWT filter to perform the 1D-DWT transform. This transform is followed by a further 1D-DWT block to achieve two dimensional coefficients. The main problem experienced by this architecture is that complex routing and incomplete data bus utilisation are experienced. Complex routing results from the fact that the low pass DWT coefficients are fed back to the same memory as the original input data (the transposition memory), and requires careful consideration. Also, if data is fed back from lower octaves, it is most likely to have a higher resolution than input data, hence requiring a wider data bus. However, since this data bus is the input data bus then extra bits located here will be unused for part of the time. In other words hardware utilisation is not effective when this architecture is used.

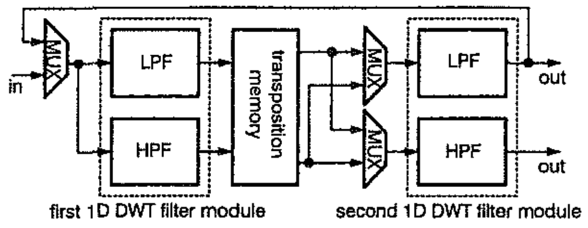


Fig 5.5

Recursive Architecture

## 5.6 RECOMMENDED ARCHITECTURE

Several Authors [40-43] have identified the Multiply-Accumulate (MAC) operation as the kernel of various digital signal processing algorithms. All the architectures discussed have used a MAC (or more) in some way. Furthermore, it can be shown that a Finite Impulse Filter (FIR) is nothing more than a collection of MAC cells connected in some way [42]. When using the recursive approach, the 2D-DWT transform can be reduced to a combination of 1D-DWT transform along the row data of the image. The resulting data is then transformed column wise along a further 1D-DWT.

The 1D-DWT transform can be loosely treated as a filtering operation in which data is passed through a high pass filter then low pass filter network. The low pass filter output is then further pass through a high pass, low pass filter network. In terms of data handling, the transition from row processing to column processing, can be realised by use of a transposition memory or by scheduling methods in which data path control is used to achieve the transposition of rows to columns.

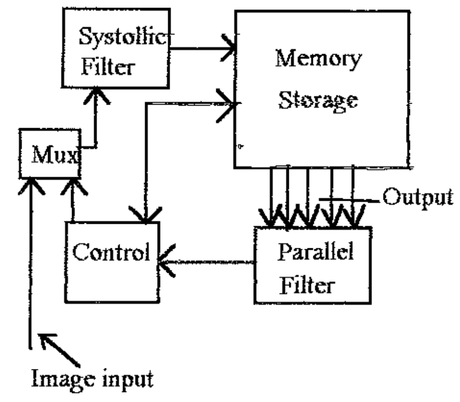
Finally, it is important to prepare a specifications list for the proposed architecture. The recommended architecture will attempt to satisfy the specifications in the best possible way. However, previous research indicates that a 'win-loose' situation is likely to be the most common outcome.

Proposed Architecture specifications list

1. Modular.
2. Simple.
3. Yields good reconstructed image quality (compared to original).
4. Fast computation time, may be used in real life applications .
5. Must fit overall DFS-EZW coding system..
6. Must have small die size.

A suitable architecture block diagram for the proposed model is shown in figure 5.6. The architecture looks very similar to Vishwanath Systolic-Parallel architecture [3]. The systollic-parallel architecture has the main property of dispersing higher level computations amongst the lowest level computation.

Vishwanath employs two 1D systolic modules to process the row data as it enters the architecture in a serial fashion. This results in 1D-DWT coefficients which are stored in a memory bank, the size of this memory bank is dependent on the filter width. From this memory bank, parallel filters perform filter steps of the column operations, the results of which are connected to the routing network in the 1D systolic modules to interleave the higher level computations onto the filtering arrays. Finally, results are available on a serial form at the output. Vishwanaths's systollic-parallel architecture is shown in figure 5.6a for comparison purposes.



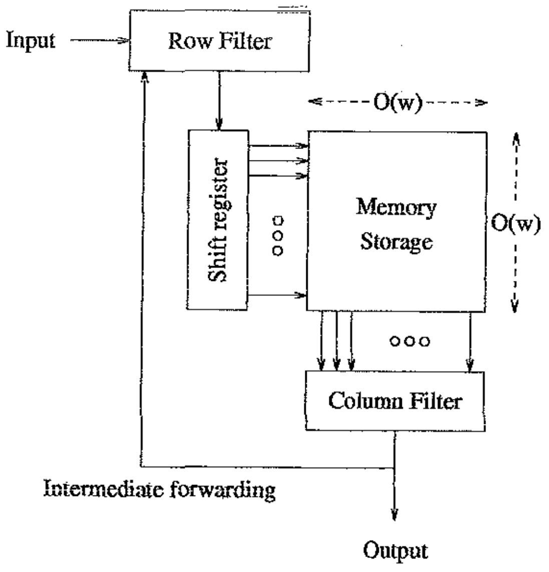


Fig. 5.6

Proposed Architecture

Fig. 5.6a

Systollic-Parallel architecture

The proposed architecture is an improvement over the systollic-array architecture. First, the architecture employs a single systolic filter and single parallel filter to perform the 2D-DWT. This is due to the fact that  filters are designed so that low as well as high pass coefficients are calculated in the same clock cycle. This may require higher scheduling but the savings on cost, which also relate to die size, are far more significant.

Next, the filters are designed using a modular approach.. This means that for instance if the target device, i.e. FPGA, capacity increases then the filter length can easily be increased without redesigning the whole system from scratch. The systollic-pararell architecture uses a filter size dependent memory storage area. In this design, memory dependency on either, image size, and/or filter length is also an unfortunate disadvantage.

The proposed architecture is also designed to fit the DFS EZW system, unlike the systollic-parallel model which fits a general 2D-DWT architecture. 2D-DWT data is generated according to the Depth First Search strategy. DFS Bitstream Results are available in parallel to individual parallel EZW processors which can then increase computational speed of the whole system.

## 5.7 XILINK XC40000

The MAC unit presented in this paper consist of a modular (see VHDL design) type MAC cell. In order to be consistent with the small die size, the word sizes must be carefully chosen to balance the size of the implementation, which is limited by the target device density.

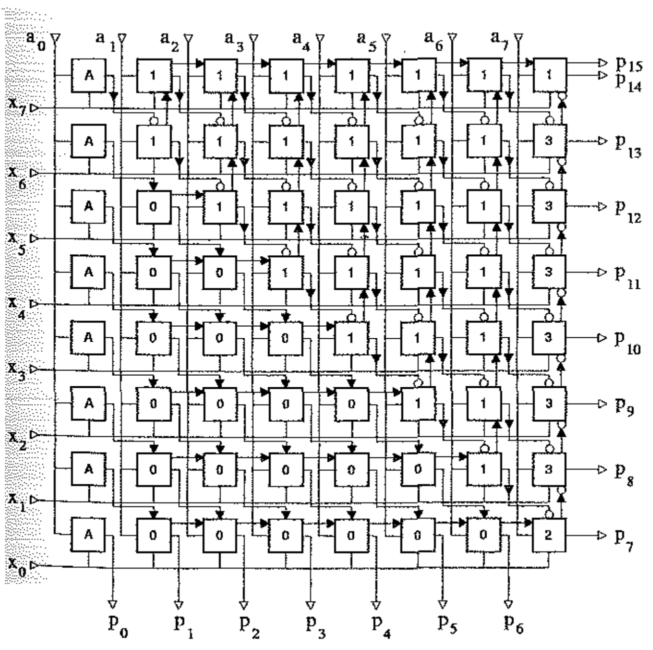
In order to consider design issues, The Xilink XC4000-series FPGA is assume to be the target device to implement the DWT architecture. The importance of chip size can be measured in terms of gates. Typically 2D-DWT achitectures employing 4-tap filters use less than 10000 gates on a target device. A Xilink XC4000 consists of an array of Configurable Logic Blocks (CLBs), each of which has several inputs (F1-F4, G1-G4) and outputs (X<Y and XQ,YQ). Each CLB also contains both random logic, and synchronous elements in addition to the special-purpose logic functions.

The XC4000 series contains both local and global routing resources. The local resources allow extremely low delay interconnection of CLBs within the same neighbourhood, as well as more extended connection through the use of switching matrices. The global resources provide for the low-delay distribution of signals that are used at widely-spaced points in the array. The speed of a particular application is highly dependent on routing in the Xilink FPGAs. The XC4000 family includes parts ranging from 8x8 CLB arrays to 24 by 24 CLB arrays. All of these devices are in-system programmable. Low power versions of many of these parts are also available.

## 5.8 MAC IMPLEMENTATION

The basic elements in the MAC unit can be defined as a multiplier, and adder and register or delay unit. Furthermore, since the multiplier most of the time consists of adders and gates performing AND Boolean functions, it can be said that the MAC unit consists entirely of adders and registers.

The multiplier employed in most designs uses a combinational array of adders to achieve multiplication as seen in figure 5.7. Where a combinational array multiplier is used, the width of the multiplier is dependent on the precision of the data and filter coefficients to be multiplied. An eight bit data representation is typical. If the filter coefficients are also 8 bit precision, then in this case a 64 element device, requiring 64 CLBs on a FPGA is required. Furthermore Since some filter coefficients are negative, then the multiplier must be configured for signed two's complement notation by generalising the adders in the combinational array multiplier as discussed in [43]. Generalised adder configurations can be seen in figure 5.7b bellow.



$a_0$ $a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$
$x_7$ $x_6$ $x_5$ $x_4$ $x_3$ $x_2$ $x_1$ $x_0$
$p_{15}$ $p_{14}$ $p_{13}$ $p_{12}$ $p_{11}$ $p_{10}$ $p_9$ $p_8$ $p_7$
$p_0$ $p_1$ $p_2$ $p_3$ $p_4$ $p_5$ $p_6$

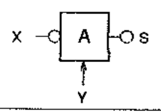
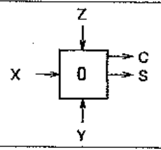
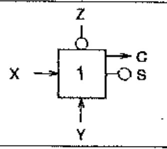
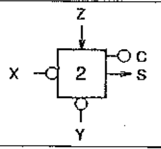
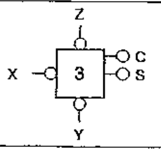
| Type | Logic Symbol | Operation |
|---|---|---|
| Type A Cell |  | $(-X) \cdot Y = (-S)$ |
| Type 0 Full Adder |  | $\begin{array}{r} X \\ Y \\ +)\ Z \\ \hline CS \end{array}$ |
| Type 1 Full Adder |  | $\begin{array}{r} X \\ Y \\ +)\ -Z \\ \hline C(-S) \end{array}$ |
| Type 2 Full Adder |  | $\begin{array}{r} -X \\ -Y \\ +)\ Z \\ \hline (-C)\ S \end{array}$ |
| Type 3 Full Adder |  | $\begin{array}{r} -X \\ -Y \\ +)\ -Z \\ \hline (-C)\ (-S) \end{array}$ |

Fig.5.7

Combinational Array Multiplier

Fig. 5.7b

Generalised full adder cells

The MAC also contains adders to summate the result of the multipliers. Multiplication of an 8 bit precision data with 8 bit filter coefficients, results in data of 16 bit wide. Careful consideration must be given to the precision of multiplication data, which can be very wide. Again research indicates that much higher precision is used, specially in the filter coefficients which can be very wide, needing many bits to be represented accurately. This results in very wide adders which may not suit single IC implementation. Further considerations to data and filter coefficients precision are given later in VHDL design.

The performance of the MAC unit with an 8-bit by 8-bit multiply and 16 bit accumulator is determined by the speed of the multiplier. The Combinational multiplier seen in figure 5.7 has a reported delay of 100 Ns [9]. Also the MAC employed in [9] can support a clock speed better than 10 MHz. With the use of the horizontal longlines to distribute the critical path signals, the speed can also be further improved, although this may restrict the use of the MAC unit in various system configurations. The single stage of a four tap MAC unit took 73 CLBs to implement on the XLINK XC40000.

## 5.9 FIR FILTER IMPLEMENTATION

The purpose of the MAC unit is to form the structure of the Finite Impulse Response (FIR) filters used in the coding system. The transfer function of such a filter is given by:

$$H(Z)= a_0 + a_1 Z^{-1} + \ldots + a_{n-1} Z^{-(N-1)} \qquad (14)$$

The structure of the filter can be realised in many ways as seen in figure 5.8. The most common structure used is the canonical structure or inverted structure. This structure provides a simple design , data flow approach, and is suitable for achieving high sampling rate even for higher order filters.

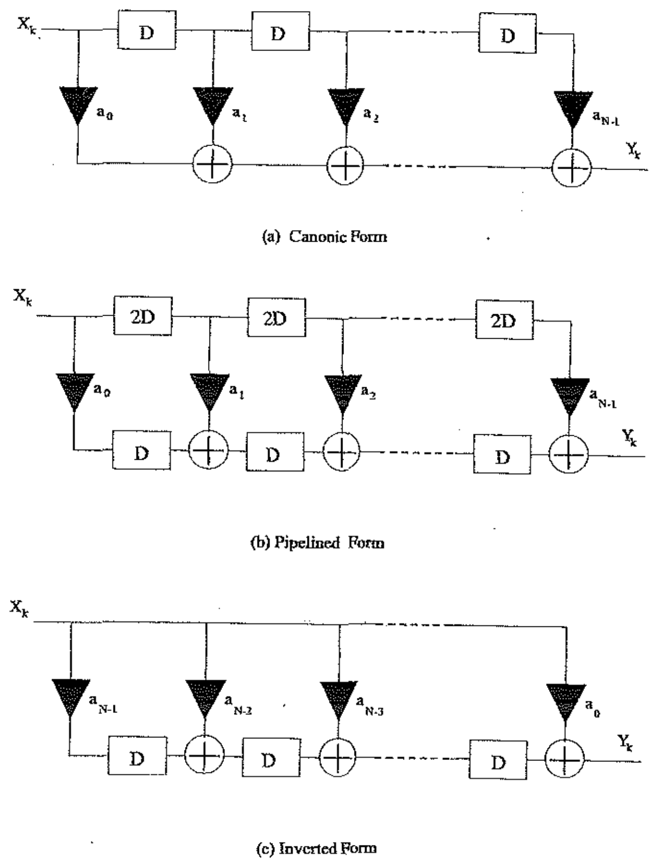


Fig.5.8

FIR Filter Structures

## 5.10 MEMORY BANK

The memory bank is the area where intermediate coefficient values are temporary stored as they are processed from individual 1D-DWT modules. Current non-separable designs, which includes the proposed architecture, employ image or filter length dependent memory banks. However, the use of memory banks is a major disadvantage of non-separable architectures because of the latency involved with data shifting between the memory and the computational units.

Many designs use shift registers as the memory storage area. Designs using this methology use efficient methods like Forward Register Allocation (FRA) or Forward-Backward Registration Allocation (FBRA) to have very small storage areas. The main advantage of these systems is that the cost associated with memory is very small, also memory can be an internal part of the whole architecture. However, this architecture can only perform the 1D-DWT, to perform the 2D-DWT requires a transposition memory to be used between individual 1D DWT units.

Using a transposition memory between 1D-DWT modules to achieve the 2D-DWT is a valid solution but does bring some problems to the performance of the architecture. First, the transposition memory will cause an increase latency. Also, the transposition memory is highly dependant on the size of the image that needs to be transformed and thus is not suitable for transforming images of an arbitrary size.

The proposed architecture also uses a memory as a storage area where intermediate coefficients are located. To attempt to produce an architecture that can process an arbitrary image size, then a large external SRAM memory is proposed. The static ram has low cost and can have a large storage area. Access to read and write to this memory is also very fast and so latency is reduced.

The process to perform the 2D-DWT transform can now loosely be defined as that of MAC units performing one dimensional wavelet coefficients These Coefficients are first processed row by row by the systolic filter and fed to the memory bank by the control unit. Next, the coefficients are process column by column by the parallel filter. 2D-DWT data from the parallel filter is finally stored in the memory bank by the control unit. Finally, lower frequency transformed data can be fed back for lower level decomposition as required.

## 5.11 CONTROL UNIT

The proposed architecture have chosen systolic filters because these filter require no control, hence simplify the operation of the architecture. However, the use of a memory storage area between the systolic filters requires that scheduling is carefully devised so that no errors are produced in the wavelet coefficients. The purpose of the Control unit is therefore to control the data path of the architecture. Furthermore, the control unit receives signals (see VHDL design) which allows it to set the memory storage area accordingly and hence process images of fairly arbitrary size. Also, the control unit redirects data according to the decomposition level applied or desired. Finally, data streams following the DFS strategy are made available in parallel by the control unit.

# CHAPTER 6
# VHDL IMPLEMENTATION

## 6.0 VHDL FOR HARDWARE DESIGN

VHDL stands for Very High Speed Hard Ware Description Language and is a powerful tool for functional verification and commercially available logic synthesis tool for synthesis verification[44]. A suitable architecture has been proposed for implementing the 2D-DWT, it is desired to verify operation of the proposed architecture and to access real time applications performance. Similar architectures have been synthesised in VHDL [3], these show performance capable of real time applications where fast computational time is required. The proposed architecture employs only two systolic filters and a memory storage area, which in conjunction with a control unit can achieve the 2D transform.

The objective of VHDL implementation is to verify operation of the proposed architecture. Since functionality is the main concern, the structures that employ the architecture are described, in VHDL, at more of a behavioural level than a structural one. However, structural models are also given consideration as these ultimately comprise the synthesisable model. Further research opportunities are then provided to finalise the structural model which requires much timing issues to be resolved.

The basic unit for a VHDL description is the design entity. An entity in VHDL may describe a system at different levels. For example, it may model some combinational logic with a set of Boolean equations, or contain an abstract description of a whole system. Entities can then be connected together allowing complex designs to be broken down into simpler blocks.

A VHDL entity, as seen in figure 6.1, can be described in two parts. The first part is an entity declaration that describes the inputs and output of the entity. The second part is an architecture body that describes what goes on between the input and output ports..

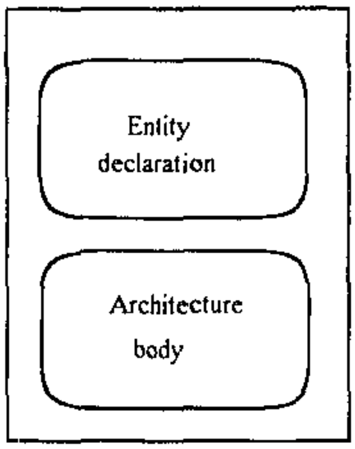


Fig.6.1
VHDL Design entity

VHDL is a powerful development software and allows designs to be described at different levels. In practice, RTL code or synthesisable code can be produced even from the most abstract or at the highest level of development. However, structural written VHDL code matches more accurately the final circuit operation. Timing operations and data path control are the most important aspects occurring at structural written code.

Code written at the top level is written in a very abstract way and includes the system description in terms of what it does. Typical entities written at this stage may include a Microprocessor and the inputs and outputs to this system. Usually, no consideration is given at this level to the form of input and output, hence it can be real or integer numbers.

## 6.1 HARDWARE FOR IMPLEMENTATION

The hardware chosen to implement the DWT transform is the Field Programmable Gate Array (FPGA) which has a widespread use in the development of Application Specific Integrated Circuits (ASICs). FPGAs are used in what is called 'semi-custom' hardware design. The main advantage of the use of FPGAs in ASIC logic circuit design is that they make circuit fabrication much quicker because all the necessary hardware is already fabricated in silicon. FPGAs are prefabricated integrated circuits containing sets of logic blocks, which contain gates, flip-flops, multiplexers and RAM, etc. This ready-made hardware is then 'wired together' by the designer to create the desired digital logic circuit.

The integrated circuit making the FPGA contains Configurable Logic Blocks (CLBs) and input/output (I/O) blocks. Each configurable logic block contains flip-flops and logic gates. The configurable Logic Blocks are then connected to form desired logic which is then connected to the I/O pins. Connections are made by programming the connections into the FPGA's own internal memory that then controls the switching matrices.

## 6.2 CONTROL

All scheduling of the proposed architecture is performed by a control unit which takes care of the data path and scheduling required by the DWT process. Since the proposed architecture uses systolic filters (see filter design), then no control requirement is required for the filters. It is for this reason that systolic filters are commonly used for implementation of the 2D-DWT.

The main job of the control unit is then simplified to the task of delivering the correct data which could be either input data, intermediate DWT data or output data to the right location for processing. In order to simplify design of the control unit an image of size N x N is considered as it is exposed to the proposed architecture.

First, it is assumed that this image is to be decomposed by the 2D-DWT transform and that the image is square, i.e. it has the same number of rows as columns. The image is 'Clocked' through the first systolic filter, again this requires no control as the filter produces a result for every clock cycle . After $N^2$ cycles, the image has been through the first systolic filter and a 1D-DWT transform of the image is placed in the memory bank by the control unit. Control lines are available, row and column, in the control unit that determine the memory address space required. The image located in memory after being through the systolic filter and place in the memory bank consists of low and high pass coefficient data as seen in figure 6.2.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |
| l | h | l | h | l | h | l | h |

Fig.6.2.

Systolic filter output to the memory bank

According to the waveletting process, rows of the image are first passed through a low and high pass filter. This has been just achieved by passing the rows of the image through the systolic filter in the architecture. The image in the memory bank now consist of low and high pass coefficients. It is now required that the low pass and high pass filter coefficients are applied column wise to the systolic filter to achieve the first level 2D-DWT transform.

Consider how the image is stored in memory after the first row transformation took place, since the image is size N x N then memory addresses N to N would contain the transformed image. If memory was to be mapped at this stage, the following contents will be revealed.

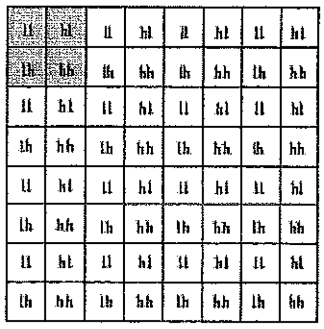
2K+1 for K= 0,,1,2,3..K-1 contains all the high pass coefficients

2K for K=0,1,2,3,4,5....K-1 contains all the low pass coefficients.

From this observation, the first level control algorithm can be derived;

```
While not done ( A bit sent after N x N Clock cycles, hence image transfer)
{
Let K=0 (initialise memory pointer)
On positive clock cycle ( Low pass filter of systolic filter takes place at this stage)
        Memory write to location 2K
         increment K
On negative clock cycle( High pass filter of systolic filter takes place at this stage)
        Memory write to location 2K+1
         increment K
 }
```

Secondly, columns of the transposed row image need to be process for the 2D-DWT transform. This takes place in parallel by the second parallel filter which is also a systolic filter. The control unit has to ensure the correct transformed columns are send to the parallel filter for processing and store the intermediate result back to memory. The result of this operation is the first level 2D-DWT. The parallel filter processes the high and low pass filter components of the row transformed coefficients in a column fashion and the control unit writes the results of this back to memory. The resultant image in memory now consists of two dimensional low pass and high pass coefficient data as seen in figure 6.3.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ll | hl | ll | hl | ll | hl | ll | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| ll | hl | ll | hl | ll | hl | ll | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| ll | hl | ll | hl | ll | hl | ll | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| ll | hl | ll | hl | ll | hl | ll | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |

Fig. 6.3.
First level, two dimensional coefficient data in memory.

More importantly, the original row transformed image located in memory space N x N is replaced by the first level 2D-DWT coefficient data generated by the parallel filter. This scheduling control is very important as otherwise very large memory is required for higher level decompositions.

The task of the control unit is hence to send both the high pass and low pass row filter coefficients to the parallel filter. Concurrently executing processes are possible in VHDL making this an ideal tool in development of parallel circuits.

Low and High pass row transformed coefficients can be then transformed by the parallel filter to achieve 2D-DWT as already discussed. A suitable algorithm can also be developed by examining the location of the low and high pass coefficients in the memory bank;

```
For all elements in row transformed image ( N x N)
  On positive clock cycle (required to synchronise process)
  When done ( image was transformed and placed in memory)
  for m=0 to Columns -1 increment 2  ( for all columns containing low pass
  coefficients)
        for n=0 to Row-1 increment 1 ( for all elements in odd columns)
        memory read n + Column x  m
  On negative edge ( parallel filter can process one 2D-DWT coefficient per cycle)
        memory write 2D-DWT coefficient ( and repeat for loop for other elements)
```

Resuming the process so far, a N x N image is row transformed by a systolic filter and the resultant row coefficients are written to memory. The row coefficients are then column transformed by a parallel fitter and the first level 2D-DWT is written to memory. Further level decomposition, involves exposing the low pass 2D-DWT coefficients to the same procedure.

It has been seen that Mallat's implementation of the 2D-DWT involves a recursive pyramid where the low pass DWT coefficients are further broken down by further decomposing them. In this design Mallat's recursive pyramid to implement the 2D-DWT is also followed. The reason for this is that Mallat's recursive pyramid is a computational effective way to implement the dimensional transform.

The function of the control unit for higher level decompositions is to control the data path so that the low pass 2D-DWT coefficients are again send to the transform process. This is equivalent to the recursive processing of the low pass components in Mallat's algorithm. Again, with reference to figure 6.3, the location of the low pass coefficients in the image in memory can be used to develop a suitable algorithm;

```
For all elements stored in memory N x N space
 when done
    For m=0 to row-1 increment m by 2 (for all even rows)
      {
          for n=0 to column - 1 increment n by 2 ( for all even columns)
           memory read address =  n + column x m ( send data to row filer )
      }
     Write result of row transformed 2D-DWT to memory.
```

Not all coefficient elements are transformed by this algorithms. At the second level of decomposition, only odd rows and columns of the transformed image are required to be processed since these contain the low pass coefficient data required. In other words, only the low pass coefficients of the first level decomposition are send to the row filter for further decomposition. Also, because not all first level coefficients are affected by the above algorithm ( only odd columns and rows) then some coefficients of the original first level transformed image remain unaffected by the above algorithm. The resulting image located in memory consists of a row transformed 2D-DWT coefficient data as seen in figure 6.4.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| lll | hl | llh | hl | lll | | llh | |
| lh | hh | lh | hh | | | | |
| lll | hl | llh | hl | lll | | llh | |
| lh | hh | lh | hh | | | | |
| lll | hl | llh | hl | lll | | llh | |
| | | | | | | | |
| lll | | llh | | lll | | llh | |
| | | | | | | | |

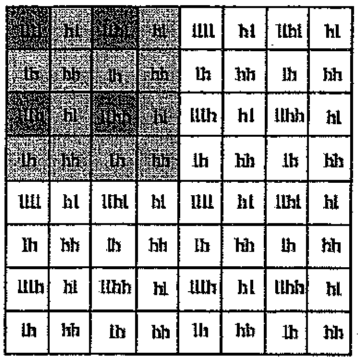
Fig.6.4
Low pass coefficients resulting from the first level 2D-DWT are row transformed by the systolic filter.

The above process has performed the 1D-DWT to level one 2D-DWT coefficients by filtering the low pass two dimensional coefficients. In order to achieve 2D-DWT coefficients at the second level of decomposition, columns of the row transformed 2D-DWT coefficients need to be processed accordingly. Again, the low and high pass coefficients of the row transformed data at the first level of decomposition is processed column wise by the parallel filter. This results in level 2 2D-DWT coefficient data produced. The control unit is again responsible to control the data path so that the right coefficients are delivered to the parallel filter. Finally, results are written to the proper location in memory by the control unit.

The same process of low and high pass filtering can be applied to existing octave coefficients. Each level breaks down the low pass filter coefficients by applying Mallat's pyramid algorithm. Again, figure 6.4 can be used as a base to develop a suitable algorithm;

```
For all 2D-DWT first level elements
  when done
  {
   for m=0 to column - 1 increment m by 4
       for n=0 to row - 1 increment n by 2
       memory read address = m + column x J ( low pass coefficient)
   on negative clock cycle
   }
 write result of column transformed (2 level) to memory

  For all 2D-DWT coefficients first level elements
   {
     For m=2 to column -1 increment m by 4
          For n=2 to row -1 increment n by 2
               memory read address = m + column x J (LLH elements to parallel
               filter)
     }
   write result to memory ( these results, together with the LLL results are the
   2D-DWT coefficients at the second  level of decomposition )
```

The decomposed image now contains second octave 2D-DWT coefficients as seen in Figure 6.5. Further decompositions are possible, each decomposed image containing fewer and fewer bits representing the low pass wavelet coefficients.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| llll | hl | llhl | hl | llll | hl | llhl | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| lllh | hl | llhh | hl | lllh | hl | llhh | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| llll | hl | llhl | hl | llll | hl | llhl | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |
| lllh | hl | llhh | hl | lllh | hl | llhh | hl |
| lh | hh | lh | hh | lh | hh | lh | hh |

Fig.6.5
2D DWT coefficients when a second level decomposition is involved.

The process of iterating the low pass filter coefficient for further octave breakdown is termed dilation [6]. Theoretically, the process can be iterated infinitely with each level producing low pass coefficients which can be further decomposed. However, if N data elements are available and $N = 2^D$ then a maximum of D dilations or decomposition levels can be applied to the image.

## 6.3 VHDL ARCHITECTURE IMPLEMENTATION

In this section the proposed architecture is implemented in VHDL using Synopsis and PeakVHDL software. During the design process, the top-down design method is followed . In doing this, two models of the system are created and simulated using VHDL. The first model as seen in figure 6.5, is an abstract behavioural model. This model is intended as an operational model more than a synthesis model. An operational model is referred to as the model where the architecture is considered as a black box or a single operational entity. Several inputs such as data, filter coefficients, control lines, etc. are used without much consideration to their format. These inputs work with the entity to produce wavelet coefficients which are then fed to further stages of the coding system.

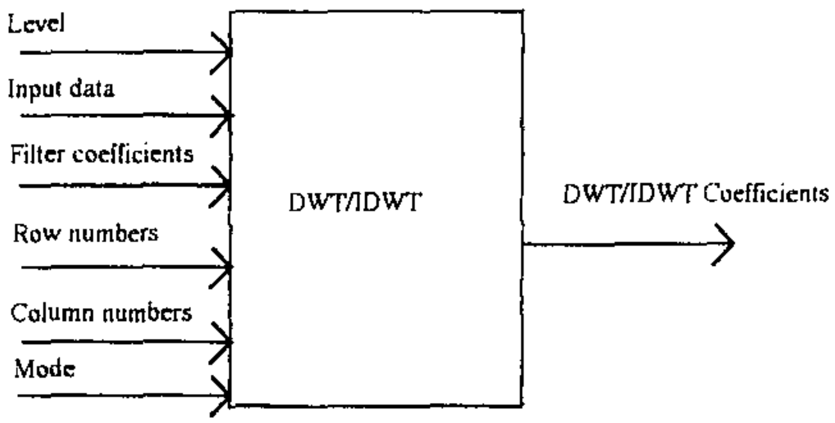


Fig. 6.5

Behavioural 2D DWT VHDL code block diagram.

As indicated previously VHDL is a very powerful language, even code at this abstract behavioural level can be translated to synthesis type RTL code. Providing that the right libraries and translation tools, such as VHDL to Behaviour Block Intermediate Format (VHDL2BBIF) are/available, synthesis is possible from behavioural code. However, These translation tools can be very expensive, even Synopsys, one of the most powerful VHDL compilers, does not have this translating tool as a basic option and requires a very expensive upgrade to be able to use it. When we consider the basic Synopsys options are already expensive by themselves, then this option could be out of economic reach. Appendix a, has a guide showing how these tools could be used to translate VHDL behavioural code to RTL synthesisable code. The tool guide could be handy when the price of translating tools decreases and these become more predominant.

The purpose of the behavioural code model is mainly to define the second structural model parameters. Data precision, which determines width of multipliers, adders, and memory storage capacity, optimal filter length, and optimal data path control are some of the useful parameters that can be obtained from the behavioural model. Structural VDHL code can be quite easily translated to Register Transfer level code which is then synthesised into the target device. Hopefully, the structural model will already be optimised with the most efficient parameters found in the behavioural model.

## 6.4 BEHAVIOURAL VHDL CODE

The first behavioural model as seen in figure 6.5 consists of a single entity. In reality this entity can be seen to consist of several working functions. Each function theoretically could be separated into an individual entity. Indeed, the splitting of individual functions into modular entities is the purpose of the structural model. The behavioural model consists of a Test bench used to test the architecture, a 2D-DWT/IDWT block and a global block. VHDL listings are included in the appendix a as indicated bellow;

| | |
|---|---|
| 1. IDWT/DWT.VHD | The VHDL 2D IDWT/DWT Behavioural code. |
| 2. TB_DWT/IDWT.VHD | The Test bench used to test the behavioural code. |
| 3. GLOBAL.VHD | The package containing global parameters. |

The above blocks are linked together, as indicated by figure 6.6., to form the behavioural test system.

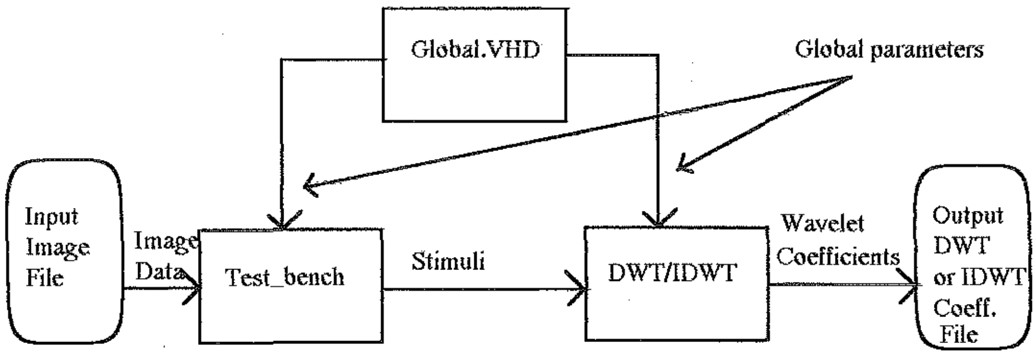


Fig. 6.6

First VHDL behavioural model.

Moreover, the behavioural model is used to determine structural parameters such as data precision and filter length. To determine these parameters, each individual block in the behavioural architecture is considered separately.

GLOBAL.VHD

A package that contains amongst other things global variables. This variables determine properties such as the size of the memory bank and other temporary variables used intermediate result storage. At this stage precision of filter coefficients and data precision has not been considered. Instead, the Real VHDL type is used for precision of data. However, the Real VHDL type has a precision of $2^n$ where n is the CPU register width i.e. a 16 bit machine has $n = 16$. The precision available when using Real data types is always unrealistic for many single chip designs. The results of multiplications, in particular, can be twice the value of the CPU register. Large multipliers may cause the design to be deployed in multi-chip environment, complicating the design because of the additional handshaking and scheduling required.

On the other hand implementing the first behavioural model using real data types simplifies the design process, results can be easily interpreted and evaluated. An interesting feature of VHDL is that it allows precision of real data types to be adjusted accordingly. This is done exactly in the same way as formatting a real type in C or C++. The width of precision of a real type can be adjusted by including the %n.n parameter after variable declaration. This gives a very powerful concept, being able to work with easy to interpret real variables, and adjusting the precision of the data type accordingly. Results can be then obtained from experiments which will give indication as to how precision of data affects image quality. Also an optimal precision parameter for the structural model will be determined from experimental results.

TB_DWT/IDWT.VHD

A standard test bench is created to test the behavioural 2D-DWT architecture. The problem with testing image coding architectures is that it must be applied to an image. Most standard images have huge amounts of data, testing for all data in the image becomes a long, tedious process. The standard test bench, as seen in figure 6.7, applies stimuli data to the Unit Under Test (UUT). Responses to these stimuli can then be accessed for evaluation purposes. However, as already stated, a standard image contains a huge amount of stimuli. Evaluation of every possible stimuli for responses would then take a huge amount of time, which is not practical.

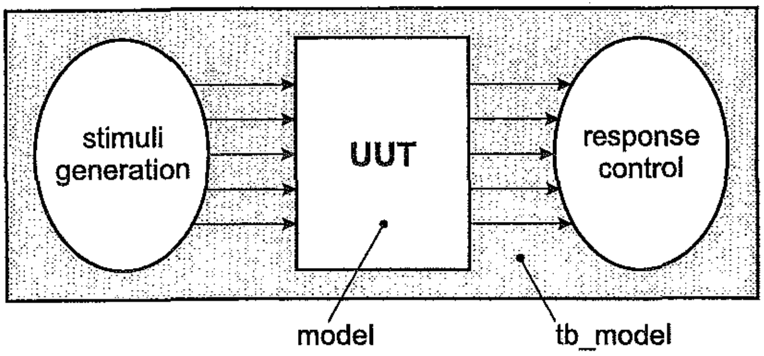


Fig. 6.7

Standard Testing of stimuli

Clearly a more practical approach is required to test architectures which work on huge amount of data. A second model of testing uses VHDL ability to handle files. Also, this model involves usage of Matlab to evaluate PSNR, a image quality measurement for reconstructed images. The second test model is an invaluable tool for testing architectures which handle huge amount of data. It is simplistic and results are easily evaluated for errors using the Matlab software.

The second test model, as seen in figure 6.8, is very simple. A file consisting of the image data is applied to the UUT. Other stimuli applied includes analysis/synthesis filter type coefficients, level of decomposition to be applied, and a synthesis or analysis mode Output of the test system is a text file which is the n level decomposition of the image, where n is the level applied in the stimuli. Matlab could be used at this stage to examine the decomposed image, this would give early indication that the decomposition stage of the DWT transform works effectively.

A better solution is to apply the n level decomposed image back to the DWT/IDWT architecture. The synthesis stimuli can be then selected for synthesis mode, indicating image reconstruction. The resultant Image file can be examined with Matlab, image quality measurements for the reconstructed image can be easily determined.. Using this approach a more efficient testing method is achieved, the full decomposition / reconstruction functionality of the whole architecture is tested for functionality.

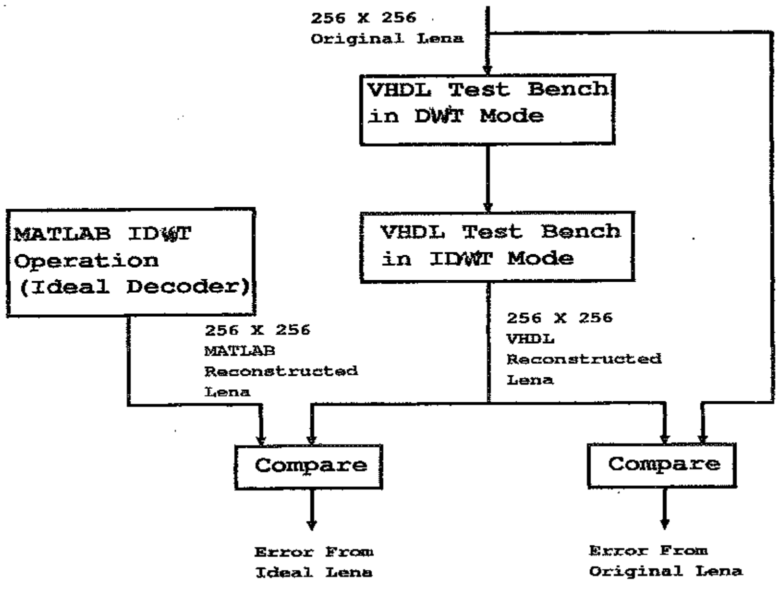

256 X 256
Original Lena
VHDL Test Bench in DWT Mode
VHDL Test Bench in IDWT Mode
MATLAB IDWT Operation (Ideal Decoder)
256 X 256 MATLAB Reconstructed Lena
256 X 256 VHDL Reconstructed Lena
Compare
Compare
Error From Ideal Lena
Error From Original Lena

Fig.6.8

Testing environment

DWT/IDWT.VHD

The main functional component of the behavioural model is the DWT/IDWT block. This block is capable of implementing the 2D-DWT and its inverse to an image of arbitrary size and to a given level of decomposition and reconstruction. Although this is implemented by a single entity three functional blocks are evident; A computational unit, A memory unit, and a control unit as seen in figure 6.9.

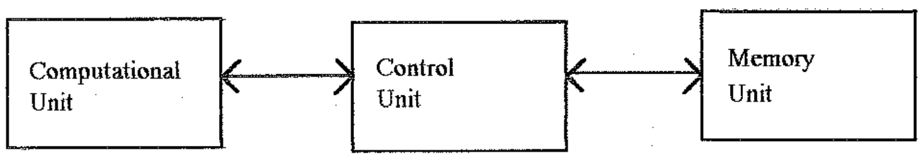


Fig. 6.9
DWT/IDWT Functional Blocks.

The Computational unit is responsible to achieve the Multiply and Accumulate functionality for the architecture. Furthermore, the function of the Low and High pass filters has been shown to consist of MAC operations . Control of the data path as well as memory scheduling control are achieved by the control unit. The memory unit is a generic, simple storage area. Intermediate coefficients are stored in this area in a two dimensional array. The length of the array is determined by parameters defined in the global package. Using a generic memory is advantageous since it allows processing of arbitrary image sizes.

## 6.5 FILTER VARIATIONS

It has been seen how important filter coefficients are in determining the accuracy of the wavelet coefficients. It is advantageous to have an architecture that permits several filters to be employed in the same design and hence change quality of wavelet coefficients accordingly. For instance, certain filter coefficients produce less aliasing than others on images.

The first behavioural model is design to produce an architecture that is capable of selecting between different filter coefficient selections. Experiments can be done on images to see the 'best performing' filter. Several of the most common filter coefficients, including the HARR wavelet filter coefficients, and some of the most common Daubechies wavelet filter coefficients are included in the first proposed architecture. It is hoped that selection of an optimal filter for the structural model can be done by experiments on test images.

## 6.6 STRUCTURAL ARCHITECTURE

The Structural model is closer to the synthesisable model or RTL model. In this model, the system is thought in terms of registers and how the data flows between them. Factors considered at this level were how the data flows between the processes, how the input data should be fed to the system, and how the results should be outputted. More importantly this model will take into account some of the constraints given by the target hardware and time becomes an essential parameter.

The system functions can be summarised as a Multiply Accumulate operation or computation stage, and a recursive feeding back of low pass coefficient data to the process as seen in figure 6.9. In the first behavioural model, these system functions are not well defined as individual entities. On the other hand in  the structural model the entities are well defined and interaction between them is critical for the operation.

The most critical parameter of consideration when designing at the structural level is time. It is required to know exactly what happens at every clock cycle. This is typical of a structural type design, we have considered WHAT is the system to do at the behavioural design and now consider HOW , which indicates a time element, this is going to be achieved.

## 6.7 MODULARITY

Many designers of 2D-DWT architectures [36-39] have failed to implement modularity into their designs. In the proposed architecture, modularity is an essential issue. Technology is a dynamic area were many changes take place. The number of transistors integrated on a single chip is growing very fast [42], from 256 thousand back in 1985 to around 2 million in 1997 and is set to continue growing. This means that density in target devices (FPGAs), grows significantly with time.

It is known that when implementing the 2D-DWT and its inverse, the reconstructed image quality is highly related to the filter length [23]. The objective is to produce an architecture which can easily be adopted to current technology and also addresses the filter length constraints. To achieve this results, the structural computational unit is designed using modular functional cells which can easily be linked together to increase filter length.

## 6.7 STRUCTURAL COMPUTATIONAL UNIT

A parallel and systolic filter are the main components of the computational unit as seen before. The structural computational unit has functional cells which implements the filter functions. These functional cells can be linked together to form longer structures, increasing the number of filter taps accordingly. Three elements are evident in each functional cell. Futhermore, beacuse both filters are systolic, each cell consists of the same elements, being;

. A multiplier.
. An adder.
. A multiplexer.
. Delay unit (D-Flip Flop)

## 6.71 STRUCTURAL MULTIPLIER

Listed as MULT.VHD in appendix c, the multiplier takes input data and multiplies it with the correct low or high pass coefficient. Data input to the multiplier can be either unsigned input data coming from the image data, or signed filter coefficient data. Multiplying a signed number with an unsigned number produces a signed result. To prevent possible errors image data is converted to a signed form and all operations are done using 2's complements form. VHDL allows standard arithmetic operations on signed numbers and produces signed results. Also, signed to unsigned conversion functions are available in VHDL. These functions can be used to make output data compatible with input image data.

Input data of n bits precision multiplied by x bits precision coefficients can produce (n+x) wide products. Some products are required to feed back to the multiplier, fed back data now has a precision of (n+x) bits, this data is multiplied by x bit filter coefficients to now have (n+x)+x bits wide precision or n + 2x. Clearly at each level of decomposition the precision of results grows bigger. Obviously because of hardware limitations, a limiting factor needs to be determined and this is where the first behavioural model is useful.

Another valid method to determine the optimal bit precision of results is to consider the worst case situation for the inputs to the multiplier in order to determine the result precision. Worst case methods highlight the maximum / minimum precision that needs to be resolved by the system. Consider the following worst case scenario:

. Input data has a range of 0-255 requiring 16 bits to be represented.
. Wavelet coefficients can have a maximum / minimum value of $\pm 1.00$. To represent this using two's complement numbers requires 1 bit for sign, 1 bit for integer part, and n bits for fraction part

Clearly, the multiplier can have a worst case situation of 16+(2+n). It is the purpose of the first behavioural model to find the value of n for optimal performance. This is achieved by performing different fractional coefficient precision transformations on test images. The PSRN image quality measurement can be evaluated for the different filter wavelet fraction precision.

## 6.72 THE ADDER

Listed as ADD.VHD in appendix c, the adder is capable of adding two 2's complement numbers and producing a result also in 2s complement form. A standard test image has been applied to the wavelet transform in the first behavioural model. Results of this simulations can be used to determine precision of the adder. The maximum output of the row and parallel filters determines the maximum precision data to be processed by the filter adders. Again, experimental data can be used to determine maximum adder precision.

## 6.73 THE MULTIPLEXER

Listed as MULT.VHD in appendix c , the multiplexer is a standard device used to direct flow of filter coefficients or data according to a control line. In practice, the multiplexer is not necessary as the control unit discussed previously can also achieve the same result. However, in order to simplify the design simple control is required. Also multiplexers used in this architecture require no control as they operate using the system clock.

## 6.74 THE DELAY UNIT

Listed as DFF.VHD in appendix c, the standard D type flip flop is used to achieve delays in data transfer. Delays are common structures used when Mallat's pyramid for the wavelet transform is used as the method of waveletting.

## 6.75 THE MEMORY

Listed as RAM.VHD in appendix c, the memory block is a generic memory structure. Four lines implement the memory operation, as seen in figure 6.10 bellow. Address, data and control lines determine behaviour of the memory which can be simplified to the following procedure;

. When the read line is active, data in the data bus is copied to the memory address specified by the address bus.
. When the write line is active, data in the specified address is copied to the data bus.
. An enable / disable line is included to prevent read and write operations occurring at the same time.

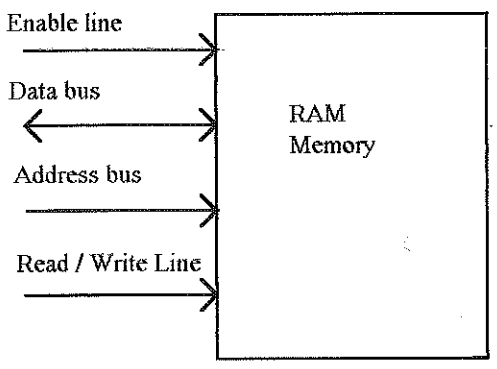


Fig.6.10

Generic RAM block

**6.76 STRUCTURAL CONTROL**

The purpose of the structural control is to provide the data path for the architecture as well as memory scheduling as explained section 6.2. Several timing issues need to be resolved before producing VHDL structural code.

# CHAPTER 7
# RESULTS

## 7.0 INTRODUCTION

A behavioural model architecture has been implemented and verified in VHDL. The standard image "Lena" is a moderate complex image that has become a standard test image for image compression techniques. It is important to note that the 2D-DWT subband decomposition does not achieve compression by itself, it merely decomposes the image into octave bands according to the level of decomposition applied. Since no image coding is achieved by the subband decomposition, the only results that can be measured are directly related to the decomposed image. In terms of results there are three areas of consideration given to this design:

. What filter gave the best performance in terms of image quality.
. What level of precision (coefficients and data) is required to achieve acceptable image quality.
. What is the throughput of the tested architecture.

image quality can easily be evaluated by using the proposed behaviour architecture to decomposed a standard image. The inverse transform is then used to reconstruct the image. Matlab can then be used to determine statistical image quality data and results can be evaluated for performance of the proposed architecture.

## 7.1 RESULTS AND OBSERVATIONS

The behavioural VHDL model has the ability of selecting between different filter types. The "Lena" image has been decomposed to level three octaves by the wavelet transform. Furthermore, the octaves are inverse transformed to obtain the reconstructed original image. Because the model is a loosy system, there will be differences between the original image and the reconstructed image. Matlab can easily evaluate for differences between the original and reconstructed images by calculating the Mean Standard Error, an efficient statistical image quality parameter.

| **Filter** | **10-18** | **13-11** | **6-10** | **9-7** | **5-3** | **2-6** | **9-3** | **2-2** |
|---|---|---|---|---|---|---|---|---|
| **MSE dB** | 26.8 | 26.1 | 29.1 | 28.9 | 41.4 | 45.6 | 33.1 | 58.6 |

Table 7.1

MSE for different wavelet filter coefficients

Table 7.1 results indicate the higher tap filter 13-11 produces the best results in terms of MSE. However, this requires 13 multipliers, 13 registers, 13 multiplexers, and 10 adders to be implemented in a parallel filter. The same amount of components are also required in the systolic filter. Clearly in terms of cost and die size, it is impractical to have a 13-11 tap filter. Shapiro's EZW algorithm originally used Antonnini's 9-7 tap filter [25]. Results from this filter yields similar results to the 13-11 tap filter but use less components in the filter.

Also, it is desired to know the affects data precision have in reconstructed image quality. Higher precision will yield better results but since there is a hardware limit to the size of the multipliers and adders which can be implemented into a single chip, an optimal value needs to be established for the structural model. In this simulation, as seen in table 2, precision of wavelet and filter coefficients for the fraction part is assumed to be the same.

| Number of bits used | 10-18 | 13-11 | 6-10 | 9-7 | 5-3 | 2-6 | 9-3 | 2-2 |
|---|---|---|---|---|---|---|---|---|
| 3 | 82.01 | 79.95 | 54.83 | 71.57 | 3.38 | 1.3 | 16.4 | 0.28 |
| 4 | 79.4 | 74.31 | 55.8 | 69.12 | 2.8 | 0.61 | 14.07 | 0.11 |
| 5 | 78.1 | 73.01 | 54.78 | 68.87 | 2.62 | 0.27 | 12.89 | 0.03 |
| 6 | 77.47 | 72.39 | 54.29 | 67.31 | 2.57 | 0.12 | 12.34 | 0.01 |
| 7 | 77.16 | 71.02 | 54.06 | 67.02 | 2.55 | 0.06 | 12.08 | 0.01 |

Table 7.2

Filter and data precision MSE dB results

Table 7.2 indicates, as expected, that higher precision results in higher quality reconstructed images. Filter length also determines how this precision affects reconstructed image quality. It is observed that for low tap filters, precision of wavelet and filter coefficients is irrelevant. The last observation is very useful, if single chip designs imply lower tap filter structures, then precision of wavelet and filter coefficients can be discarded in the design.

## 7.2 OUTCOMES

The first behavioural model serves as an experimental model. Simulation results can lead to optimal parameter definition for the second structural model. Based on simulation results, the following outcomes can be specified;

**Row systolic filter:**

Inputs:

. Input to the filter can either be data for the image or intermediate coefficient data. Since intermediate data has more precision than image data, input to the row filter should be a 16 bit, two's complement number with 1 bit sign, 8 bits integer part and 7 bits for fractional part . To achieve this unsigned image data is converted to the right format before being applied to the row filter.

. Filter coefficients can be represented by a 10 bit twos complement number consisting of 1 bit sign, 1 bit integer, and 8 bits fraction part.

Multiplier:

The structural multiplier takes a 10 bit coefficient and 16 bit data input giving a precision of 26 bits for results. However, the maximum data input and coefficient input is ( $\pm 1 \times 255$) = $\pm 255$. Hence, a 17 bit twos complement number is sufficient to represent maximum or minimum condition. Thus, results are rounded of to 17 bit precision number. Rounding off has an insignificant affect on reconstructed image quality because filter lengths are kept relatively small.

Adder:

Signed 17 bit results of the multiplier can be added from previous stages which are also 17 bit signed. A 20 bit adder should be sufficient to account for addition of two 17 bits signed data.

**Parallel filter:**

This filter is very similar to the systolic filter since it uses the same components. However, because it operates on lower octave data, higher precision input is expected. On the other hand, data output from the row systolic filter is rounded off before being placed in memory. Due to the rounding operation taking place at the row systolic filter, input data to the parallel filter has the same precision as input data to the row systolic filter. Again, due to the low filter length employed, no significant image quality degradation occurs due to this rounding of data, but the design is greatly simplified by using the same data precision through the architecture.

Finally, a further outcome from the behavioural model is the optimal filter length required for the structural model, sometimes referred to as the number of filter taps. It is desired to use longer filter taps because they yield better performance. On the other hand, very long filters may not suit a single chip design. Furthermore, despite having poor reconstructed image quality, low tap filters are independent on coefficient and filter precision to produce good results. The 4-4 tap filter configuration is an efficient, single chip, data precision independent design used successfully in wavelet architectures [39].

## 7.3 THROUGHPUT

The proposed architecture has been accessed in terms of size and quality requirements. A very important measurable quantity is the throughput of the proposed architecture. This quantity indicates how long it takes to decompose and reconstruct an image and is an useful indication of the architecture performance because real time applications such as digital video decoding / encoding require high throughput.

Throughput can easily be measured once the time to decompose or reconstruct an image is measured. Consider an image with N number of elements, to calculate the architecture's throughput for this image the following applies;

$$\text{Throughput} = \frac{N}{T} \tag{15}$$

Where N is the total number of pixels for the proposed architecture and T is the time it took to perform the simulation.

When the "Lena" image was 2D-DWT wavelet transformed to level 3 octaves, the simulation took 17212400 nanoseconds. Throughput is found by applying formula 15, with N=256 x 256 and T = 17212400 ns. The DWT part of the architecture can process around 38 MP/s or 38 million pixels per second. The decomposed image was then applied to the inverse transform and once again a throughput of around 38 MP/s was obtained. Hence the throughput of the DWT/IDWT is 38 MP/s. Real time video signal processing requires throughput higher than 30 MP/s . Clearly, the proposed architecture is suitable for real time applications.

## 7.4 LATENCY

The separable approach  makes use of 1D-DWT modules and memory storage to implement the 2D-DWT. This approach is a simple, straightforward solution which is adopted by many  2D-DWT architectures. Furthermore,  The separable approach based architectures have the advantage of simple design and produce coefficients which are well localised in time and frequency. On the other hand it is recognised [37] that latency for this systems is too long and the needed memory space used between row and column DWT processes is large causing problems for single chip designs.

Clearly,  as the level of decomposition increases the overall latency increases due to the recursive nature of the separate approach. It is for this reason that the separable approach has sometimes being defined  as non suitable for real time signal processing [17]. .

To prevent the latency-memory problems of the separable approach some authors  have proposed using non separable architectures [37]. These architectures are highly suitable for real time signal processing applications. However, the non separable approach yields wavelet coefficients which are not well localised in frequency and time. Coefficients resulting from this non separable approach  appear as blocky artifacts when the image is reconstructed much like the DCT transform produces blocky artifacts for low bit rates. Current research is taking place for removing deficiencies in non separable as well as separable methods for subband transformation.

## 7.5 COMPARISON TO OTHER ARCHITECTURES

Wavelet transform architectures form part of larger image coding systems. In terms of results, the 2D DWT/IDWT architecture is rarely tested for performance on its own. The approach taken all the time is to test the whole coding system rather than individual entities within the system. Several wavelet architectures [36-39] have been proposed and verified for performance. Performance of all architectures is measured with the same objectives; high-speed, small size and efficient designs . Some designs cover certain areas better than others. The proposed model also has its own benefits and disadvantages as seen in bellow.

| Archicteture | Folded [2] | Digit [2] | Systolic [3] | Direct [11] (processor) | systollic-parallel [11] | Parallel [13] | Sheu [4] | Propo- -sed |
|---|---|---|---|---|---|---|---|---|
| Multipliers | 16 | 14 | 24 | - | 4L | 4L | 16 | 8 |
| Adders | 14 | 12 | 24 | - | 4L | 4(L-1) | 16 | 6 |
| Registers | 164 | 258 | 48 | N | 2NL+4N | 2NL+N | 108 | 8 |
| Scheduling | Complex | Simple | Simple | Complex | Simple/Complex | Simple | Simple | Simple/ complex |
| Period | na | na | na | 4N | N+N | N+N | na | N+N |

Here ,L is the filter length, N is size of image, and na means not applicable

## 7.6 INPUT OUTPUT CONSIDERATIONS

Although no consideration has been given to input and output of data from the proposed behavioural model architecture, in the structural model these are very important if the system is to function properly. Input to the architecture consists of image data stored in a file. This data is send through a serial input in a row major form. Handshaking is essential for proper timing between input of image data and recursive feeding of higher octave data.

The raster scan method indicates the order in which subbands or wavelet coefficients flow from the wavelet architecture. These subbands are the output of the architecture to the EZW quantiser which operates on the coefficient bitstream received from the wavelet transform. The Depth First Search (DFS) approach is an alternative way to encode wavelet coefficients from the 2D-DWT architecture. From the point of view of the quantizer, the only difference is the way in which the coefficient bitstreams arrive at the quantiser.

In this design, the programming ability of the processor design architecture is employed. The reason for this is that output is easily controlled to achieve desired results. In this case, the output is a parallel bitstream of DFS coefficients. Again, there is a requirement for handshaking to occur. The wavelet transform sends coefficient bitstreams which follows the hierarchy of the tree from the root to the children in the DFS strategy. Further bitstreams should not be processed until the quantiser has finish encoding / decoding the current bitstream.

# CHAPTER 8

## 8.0 CONCLUSION

In the last decade, there has been an enormous increase in the applications of wavelets in various scientific and industrial applications . The major contribution of wavelet theory is to relate the discrete time filterbank with the theory of continuous time space. Furthermore, wavelets can been shown to have far superior features for image processing applications such as;

. Adaptive time-frequency windows
. Lower aliasing distortion for signal processing applications
. Computational complexity of O(N), where N is the number of data samples.
. Inherent scalability
. Efficient VLSI or VHDL implementation.

Since DWT requires intensive computations, in particular the 2D-DWT, several architectures have been proposed to efficiently implement this function. However, while some architectures have tried to address some issues such as speed, complexity, or cost. It is always a 'win-loose' situation. For instance, the parallel filter architectures [35] addresses computation speed but fails to address die size and cost. Other efficient architectures, such as the systolic architecture [35], achieve the 2D-DWT with very simple designs and achieve high computational speeds but fail to address latency and have large memory storage requirements.

There is a clear need for designing and implementing a DWT chipset that explores the potential of DWT particularly in the area of high computational speed and suitability for the EZW based coder. In order to achieve the requirement of high computational speed and suitability for the EZW coder the following architectures have been considered

. The Parallel architecture for its high computational speed.
. The Systolic architecture for its simplistic design features.
. The direct (processor based) architecture for its ability to adapt to multiple applications.
. The recursive architecture

The sytollic-parallel architecture [35] has been implemented and verified in VLSI. In this architecture the 2D-DWT is implemented by dispersing higher level computations amongst the lower level computations. Two one dimensional systolic modules are used to process the rows as the data enters the architecture in a serial fashion. Results of this processing are stored in a memory bank that is dependent on the filter width. From this memory bank, parallel filters perform filter steps of the column operations, the results of which are connected to the routing network in the one dimensional systolic modules to interleave the higher level computations onto the filtering array. The routing within the block memory prevents any blocking affects being introduced in the transform.

The proposed architecture looks and operates very similar to the systolic-parallel architecture. However, there are some key differences which makes it uniquely different;

. Modular filter design enables easy future filter expansion
. The proposed architecture is designed to suit the EZW based coder
. Digit type control enables output of coefficients in a raster type format or DFS format.
. Output is made available in parallel to suit DFS EZW coding system.

A suitable architecture for the EZW based coder has been presented. The architecture employs systolic-pararell filter to implement the non-separable 2D-DWT. Simulation results indicate a high throughput for the employed systolic filters. However, it was also observed that a high latency resulted from the use of a memory storage area. This latency was further aggravated by the recursive nature of the non-separable approach taken.

Results obtained in simulation experiments show a throughput of 38 MP/s for the behavioural model. However, when compared with the throughput of 100 MP/s for the lattice based architecture and separable transform type architectures, it is clear that early results indicate an unsuitable or marginal system results if the recommended architecture is used. To improve this latency situation many authors [37] have proposed using different scheduling approaches and methods such as pipelining to increase throughput.

The proposed architecture was verified by implementing a behavioural model in VHDL. The purpose of implementing an architecture in VHDL is to verify that it operates properly. After verification the architecture can be synthesised into a target device such as a FPGA.

When implementing the architecture in VHDL, the Top Level approach was followed. First, an abstract behavioural model was developed. The purpose of this model was;

. Simulate the behavioural architecture

. Define parameters and specifications for second structural architecture.

. Obtain experimental data such as optimal synthesis/analysis filter and data precision

A structural arctitecture model is also considered . This second model will be written in a structural form and will be closer to the synthesisable VHDL code. The main issue in the second structural model is time, it is required to know precisely what data movements and calculations occur, at every clock cycle.

In designing the final architecture a first behavioral model was created and certain assumptions about how the overall architecture will operate were made. The first behavioural model verified these assumptions through the use of simulation experiments. For instance, the minimum binary word lengths that can be used, which allows for some optimisation in the second structural model, are determined. Also, the first behavioural model can be used to determine the effects on image quality by using different data precision, filter type, and rounding off .

The second model is used to define the registers that will be contained within the final implementation of the system. This model will take into account how the data is taken in the system and also outputted. Optimal data precision found in the first behavioural model can then be implemented at this second model.

Results of the first behavioural model have been examined. These results indicate that the behavioural model design does implement the 2D-DWT to an arbitrary level of decomposition. Also requirements have been placed on the second structural design by analysing experimental simulation results. Furthermore, these results indicate that the architecture can be considered efficient, in terms of computation speed, image quality, and suitability for the EZW based coder. Real time applications such as digital image encoding/decoding will require that attention is paid to the high latency of the proposed system.

## 8.1 FURTHER WORK

A suitable behavioural architecture has been verified by the VHDL Synopsis tools. Results obtained from simulations indicate proper functionality of the proposed architecture. Furthermore, specifications have been obtained from the first behavioural model that will be implemented in a second structural model.

Work has been undertaken towards second structural model, basic structural functional units have been implemented and verified. Simulation results indicate that these basic structural cells operate properly. However when these cells are placed together to form a working system, much consideration must be given to time which requires to know precisely what data movements exist in the architecture at every clock cycle. Development of the control unit which schedules the data path for the architecture has not being achieve at this stage. Further work is required to complete the control unit and develop the structural model. Finally, a working structural model can be synthesised in a target device.

## 8.2 CONCLUDING STATEMENT

A suitable 2D-DWT for the EZW coder has been presented and verified in VHDL. The opportunities for further research are excellent, a structural design is to be finalised and optimal data path control can be developed to reduce latency. Image coding applications are growing sharply, the need for a more efficient and powerful image coder provides sufficient motivation for future research.

## BIBLIOGRAPHY

[1] A.Gersho and R.M. Gray, Vector Quantization and Signal Compression. Kluwer Academic Publishers, 1992.

[2] J.M. Shapiro, Embedded Image Coding Using Zerotrees of Wavelet Coefficients, IEEE Trans. Signal Processing, Vol.41, no.12, pp. 3445-3462, Dec. 1993.

[3] S.G. Mallat, Multifrequency channel decompositions of images and wavelet models, IEEE Transactions on acoustics, Speech and Signal Processing, Vol.37, no.12, pp. 2091-2110, December 1989

[4] J.Villasenor, B. Belzer and J. Liao, Wavelet filter evaluation for image compression, IEEE Transactions on image processing, vol.2, pp.1053-1060, Aug. 1995.

[5] J.Korvacevic and W. Sweldens, Interpolating filter banks and wavelets in arbitrary dimensions, tech. rep., Lucent Technologies, Murray Hill, NJ, 1997.

[6] C.Herley, Boundary filters for finite-length signals and time varying filter banks, IEEE Trans. Circuits and Systems, vol.1, pp. 391-394, May 1993.

[7] C. E. Shannon, Coding theorems for a discrete source with a fidelity criterion, in IRE Nat. Conv. Rec., vol. 4, pp. 142-163, March 1959.

[8] R.M. Gray, J.C.Kieffer, and Y.Linde, Locally optimal block quantizer design , Information and control, vol.45, pp.178-198, May 1980.

[9] Dennis Gabor, Theory of communication. J.IEE, 93:492-457,1946

[10] H.Gharavi and A.Tabatabai. Application of quadrature mirror filters to the coding of monochrome and colour images. In Proceedings ICASSP, pages 32.8.1-32.8.4,1987.

[11] John G. Daugman. complete discrete 1D Gabor Transforms by neural networks for image analysis and compression. IEEE Trans. ASSP, ASSP-36:1169-1179,1988.

[12] M.Kunt, A.Ikonomopoulos, and M.Kocher. Second generation image-coding techniques. In Proceedings IEEE, pages 549-574,1985.

[13] A.B. Watson. Efficiency of a model human image code. J.opt.Soc. Am. A, 12:2401-2417,1987

[14] P.M Cassereau, D.H. Staelin, and G.de Jager. Encoding of images based on a lapped orthogonal transform. IEEE Trans. Communications, 37(2):189-193,1988.

[15] A.Grzeszczak, M.K. Mandal, S.Panachan, VLSI Implementation of Discrete Wavelet Transform, IEEE Transactions on VLSI Systems, Vol.4, No.4, pp.421-433, Dec 1996

[16] P.Faber and H.Susse, Construction and Optimisation of Discrete Wavelets, IEEE-SP Symposium on TFTSA-94

[17] P.J.Burt, Fast Filter Transforms for image processing. Computer Graphics and Image Processing, 16:20-51,1981.

[18] R.Shafer, P.Kauff, and U. Gollz. On the application of spatio-temporal contrast sensitivity functions to HDTV. In conference on Applied Vision, pages 118-121, Optical Society of America, San Francisco, July 1989.

[19] A.Croiser, D.Esteban, and C.G.Haland. Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. In international Conference on Information Sciences and Systems, pages 443-446, Patras, August 196

[20] D.Esteban and C.Galand. Application of quadrature mirror filters to split band voice coding schemes. In proceedings ICASSP, pages 191-195, 1977

[21] Edward H. Adelson, Eero Simoncelli and Rasjesh Hingorami. Orthogonal pyramid transform for image coding. In proceedings of SPI, pages 50-58, Cambridge, MA, October 1987.

[22] M. Vetterli. Multi-dimensional sub-band coding: some theory and algorithms. Signal Processing, 6(2):97-112, February 1984.

[23] G.Strang and T.Q. Nguyen, Wavelet and Filter Banks. Wellesley, MA: Wellesley-Cambrideg Press, 1996.

[24] O.Rioul, Regular Wavelets: a discrete-time approach, IEEE Transactions on Signal Processing, vol..41, pp.3572-3579, Dec.1993.

[25] M. Antonini, M. Barlaud, and P.Mathieu, Image Coding Using Wavelet Transforms, IEEE Trans. Image Proc., vol. 1, pp. 205-220, Apr. 1992.

[26] M. Unser, Approximation power of biorthogonal wavelet expansions, IEEE Transactions on Signal Processing, vol. 44, pp. 519-527, Mar. 1996.

[27] G.Deslauries and S.Dubuc, Symmetric iterative interpolation processes, Constructive Approximation, Vol. 5, no.1, pp. 49-68, 1989.

[28] I. Balasingham and T.A Ramstad, On the relevance of regularity constraints in subband image coding, in Proc. Asilomar Conference on Signals, Systems and Computers, (pacific Grove), 1997.

[29] W.Chen and W.K. Pratt. Scene adaptive coder. IEEE Trans. communications, 32(3): 225-232, March 1984.

[30] A.S Lweis and G.Knowles, Image compression using the 2D wavelet transform, IEEE Transactions on Image Processing, vol.1, pp.244-250, April 1992.

[31] B.Harder, Embedded Subband Coding Using Wavelet, Theses, Edith Cowan University, Perth W.A, 1997

[32] H.N. Cheung, L. M. Ang and G. Alagoda, Bits Stream Architecture for the Implementation of the Improved Embedded Zerotree Wavelet Algorithm, School of Engineering and Mathematics, Edith Cowan University, Joondalup WA 6027, Australia

[33] S.Panchanathan, universal architecture for matrix transposition, IEE Procedings-E, vol. 139, No.5, Sept 1992.

[34] P.Duhamel, Fast algorithms for Discrete and Continuous Wavelet Transforms, IEEE Trans. on Information Theory, vol.38, No.2, March 1992.

[35] M.Vishwanah, R.Owens, and M.J Irwin, VLSI architectures for the Discrete wavelet transform, IEEE Trans. Circuits and systems II, Analogue and Digital Signal Processing, vol. 42, no. 5, pp. 305-316, May 1995.

[36] X. Chen, T. Zhou, Q. Zhang, W. Li and H.Min, A VLSI Architecture for Discrete Wavelet Transform, IEEE Trans. Inform. Theory, Vol.36, no 5, pp. 961-1005, sept. 1990.

[37] C.Chakrabarti and M.Vishwanath, Efficient realisations of the discrete and continuous wavelet transforms: from single chip implementations to mappings on SIMD array computers, IEEE Trans. Signal Processing. vol. 43, no.3, pp. 759-771, Mar. 1995.

[38] C.Xuyun, Z.Ting, Z.Qianling, M.Hao, 2D DWT/IDWT Processor Design for Image Coding, ASIC and Systems State-key Laboratory, Fudan University, Shanghai, China

[39] S-K. Peak and L-S Kim, 2D DWT VLSI architecture for wavelet image processing, Electronic Letters, vol.34 no.6, 19 March 1998

[40] P.R Cappello, editor. VLSI Siganl Processing. IEEE Press, 1984

[41] S.Y. Kung. VLSI Array Processors. Prentice-Hall, 1998.

[42] K.Eshreghian and D.A.Pucknell, Basic VLSI Design 3rd ed., Prentice Hall, 1994

[42] S.Y. Kung, R.E.Owen and J.G.Nash, editors. VLSI Signal Processing II, IEEE Press 1986.

[43] K.Y.Khoo, A.Kwentus, and A.N.Willson, Jr. An efficient 175 MHz programmable FIR digital filter. In IEEE Int.Symp.Circuits and Syst., pages 72-75, May 1993.

[44] Synopsys, VHDL Compiler Reference Manual, Synopsys Inc., 1994.

[45] P. Dougla, VHDL, McGraw-Hill Inc., 1994

APPENDIX A

# VHDL BEHAVIOUR TO RTL MANUAL

# Users' Guide : VHDL Behavior to FPGA Implementation

**Who should use this Manual:**

This Manual is intended to serve as a Users' Guide for synthesizing a Behavioral VHDL description onto the Xilinx FPGA Hardware. The use of the tools described in the Guide requires the basic knowledge of the various steps involved in the synthesis of a behavioral description into a hardware implementation. Also, it is assumed that the user has the basic knowledge of VHDL syntax and semantics and the use of Synopsys tools for simulation. At every step, we have included an illustrative example (tlc_example), showing the inputs and results of that step.

**Set-up:**

The following steps should be performed for any design to be synthesized using the tools described in this manual:

- Uncomment the following lines in your '.cshrc' file :
```
set _use_synopsys_tools
set _use_xilinx_tools
```
- Include the following command in your ".cshrc.local" file, in order to set up the use of Xilinx M1 tools
```
source /packages/Xilinx/setup
```
- Copy the following two files to your home directory (or make necessary changes to them if you already have them)
```
.synopsys_vss.setup
.synopsys_dc.setup
```
- Create a directory called 'synopsys_work' in your home directory
- Include the following command in your ".cshrc.local" file in order to set up the use of all the other required tools
```
set path = ($path
/home/ddel/public_html/projects/asserta/bin)
```

# Design Flow

The following are the tasks that ought to be followed in order to synthesize a behavior VHDL description onto an FPGA device. The names within parenthesis denote the tools that are used to accomplish the corresponding task.

- Behavioral Description in VHDL and RTL Correspondence
- Simulation at Behavior Level (Synopsys tools)
- Translation - Behavioral VHDL to Intermediate Format (vhdl2bbif)
- Simulation after Translation (Synopsys tools)
- The Component Library Specification
- High Level Synthesis (asserta)
- Translation - RTL Intermediate Format to VHDL (rtl2vhdl)
- Simulation at RT Level (Synopsys tools)
- Logic and Layout Synthesis (rtlvhdl2fpga)

Given below is a figure depicting the entire design flow.

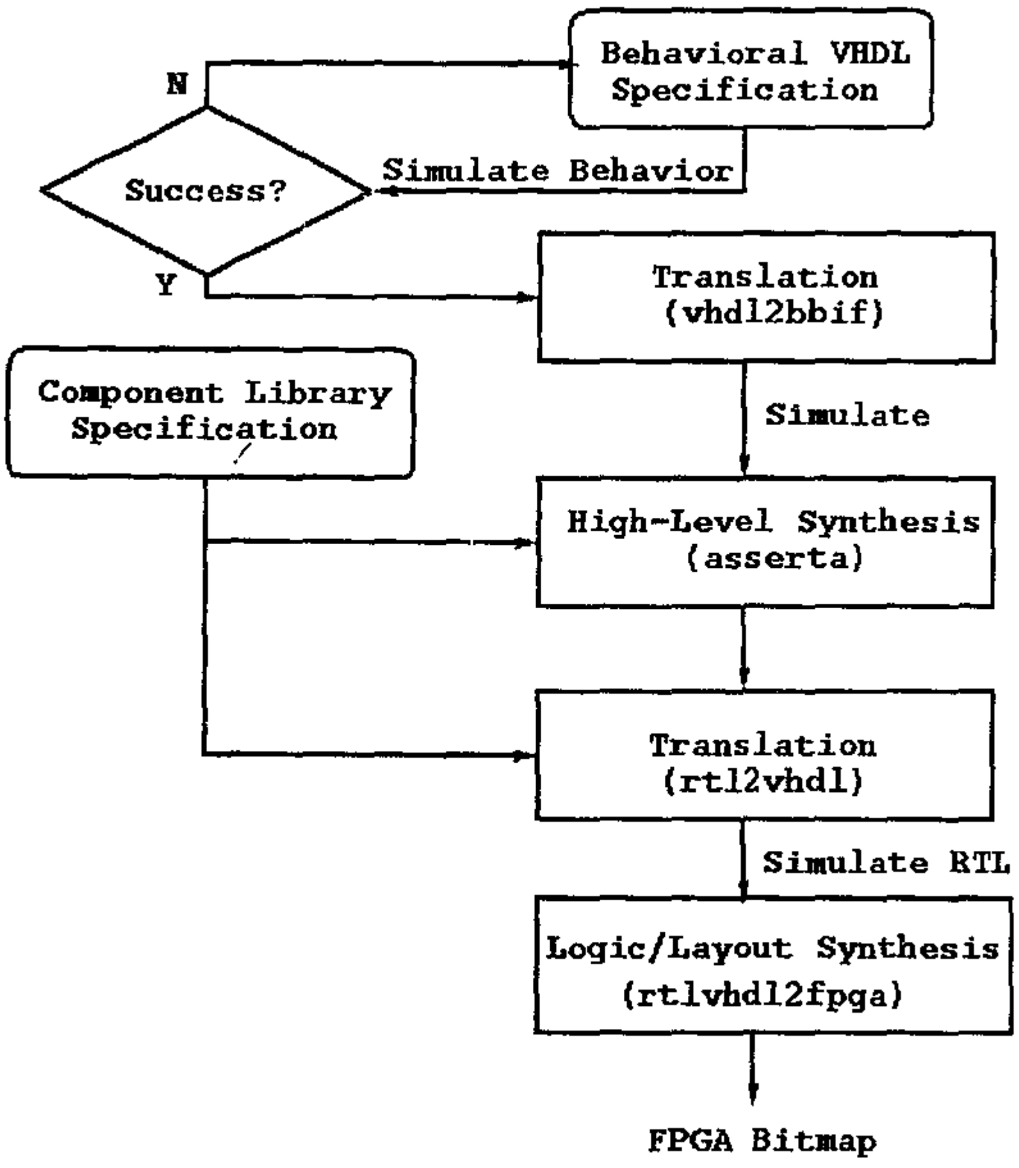

Behavioral VHDL Specification
N
Success?
Simulate Behavior
Y
Translation (vhdl2bbif)
Component Library Specification
Simulate
High-Level Synthesis (asserta)
Translation (rtl2vhdl)
Simulate RTL
Logic/Layout Synthesis (rtlvhdl2fpga)
FPGA Bitmap

Figure 1. Design Flow

## Behavior Description in VHDL and RTL Correspondence

VHDL is a language designed to describe hardware components for simulation. As such, it permits virtually unlimited number of data types and other features to model hardware behavior. The whole gamut of such features however creates problem when one thinks of synthesis. The mapping from VHDL descriptions into hardware is often confusing, when all VHDL constructs are taken into account. Moreover, some constructs in VHDL are purely for simulation.

We synthesize a VHDL (subset) behavioral description into a RTL design based on the Glushkovian model (see figure below), consisting of a data path and a controller. The datapath consists of a netlist of components picked from the component library and the controller is a finite state machine.

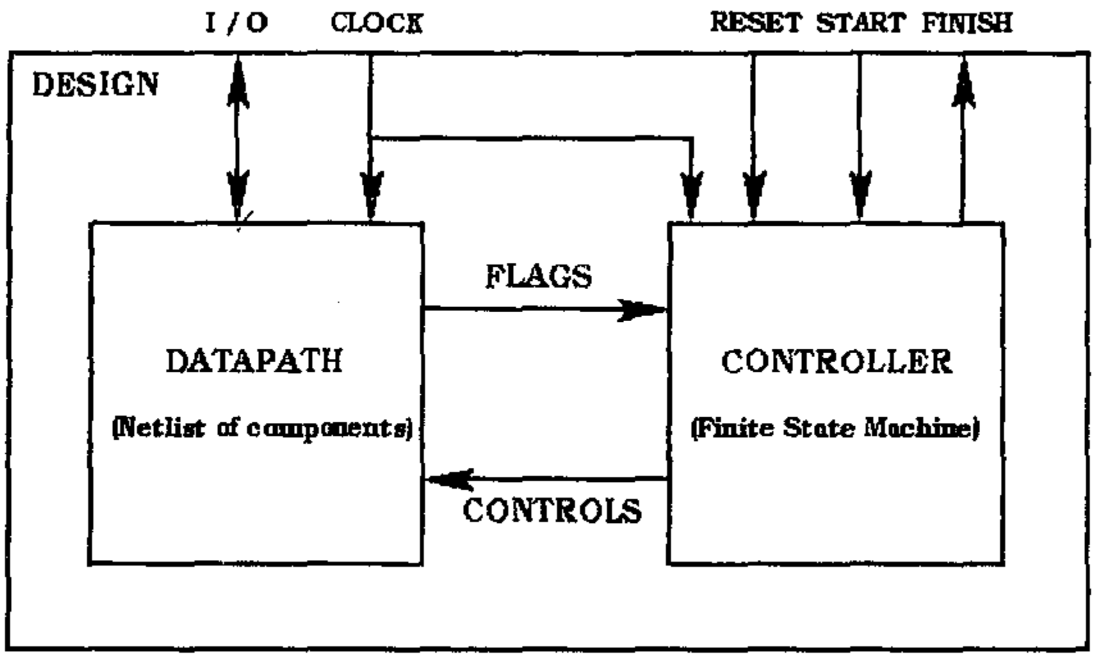


Figure 2: Glushkovian Model

The components in the datapath implement the operations specified in the behavior. The controller provides control signals that sequence these components, thereby executing the behavior specified. Note that the RTL design in addition to the design I/O ports (as specified in the behavior), also consists of four special signals: Clock (in), Reset (in), Start (in) and Finish (out). The way the RTL design comminicates to the environment is as follows. The Reset has to be made high for one clock, following which the design inputs can be placed and the Start signal is made high for one clock, indicating to the hardware that the inputs have been placed. The design then executes for several clocks and makes Finish high, inidicating to the environment that the outputs have been produced.

In the following section, we will give an overview of the VHDL subset that we propose for synthesis.

## Behavior VHDL Subset

This section provides a quick reference to the behavior vhdl subset that is supported for synthesis. It is assumed that the reader has prior knowledge about VHDL syntax and semantics. Interested readers may refer to the document "Appropriate Usage of VHDL : The Synthesis Point of View ", for a comprehensive discussion of the VHDL subset for general purpose synthesis. The following list describes the behavior VHDL subset that is supported:

- **The VHDL file** : Should consist of a single entity and architecture pair describing the design to be synthesized.
- **Entity interface** : Port declarations of mode IN and OUT are supported. Array of ports is not supported.
- **Architecture declarative part** : Signal and Constant declarations are supported. Signal initilalization is supported.
- **Architecture description** : Should consist of a single process describing the behavior of the design. In general the notion of a behavior can be defined to be an algorithmic specification of the functionality of the design. For example, if you need synthesize a sorter, you could write the bubble-sort algorithm.
- **Process declarative part** : Variable and Constant declarations are supported. Variable initialization is supported.
- **Process description** :
    - Simple Signal assignment of the form "signal_name <= expression" is supported.
    - Simple Variable assignment of the form "variable_name := expression" is supported.
    - Note : It is highly recommended for correct synthesis

that design ports be read into variables at the beginning of the process, and design ports be written to at the very end of the process.
  - The Control statements that are supported are : IF-statement, CASE-statement and the WHILE-loop.
  - Sensitivity list, Wait statements and after clauses are ignored.

- **Data Types** : The following data types are supported: "bit", "bit_vector", "std_logic", "std_logic_vector" and "integer". Note that for the integer data type, the user can provide a synthesis pragma "--$width" in order for specify an implementation width. If the pragma is not provided, a default of 16 bits is assumed for any integer. For example, the declaration
  "variable X : integer; --$width = 4"
  indicates that X should be implemented as a four bit register.
- **Arithmetic Operators** : The following two's complement operators are supported: "+", "-", "*" and "/". Note that for division the component library currently has a power of two divider. The user may use a synopsys divider to perform generic division as long as he provides it in the component library.
- **Relational Operators** : The following operators are supported: "=", ">", "<".
- **Logical Operators** : The following operators are supported: "and", "or", "not".

Example behavior VHDL Specification : Traffic Light Controller (tlc.vhd)

## Simulation of the Behavioral VHDL specification using Synopsys tools

In order to very that the design written in behavior VHDL is correct, it needs to be simulated. The next step is to write a design testbench in VHDL, for simulation. The Synopsys vhdl analyzer and vhdl simulator can be used as shown below, to simulate the behavioral description:

```
          vhdlan <design.vhd>
<testbench.vhd>
          vhdlsim
```

```
<testbench_configuration>
```

Note : IEEE synopsys library currently has a bug related to assignment statements (for example A := B + C; ). It expects the width of the output (A) to be equal to the inputs' width (B or C) in the case of + and - operators. Whereas the output width should actually be one more than the input width. Therefore for the purpose of simulation, the specification has to be remodeled using IEEE package functions, CONV_INTEGER and CONV_STD_LOGIC_VECTOR. However, for translation into the intermediate format (bbif) these inputs must be in their original form.

**Illustrative Example: TLC**

```
TestBench Specification                  :
tlc-tb.vhd
Commands used                            : vhdlan
tlc.vhd tlc_tb.vhd
                                           vhdlsim
E
Output of  Simulation process            : beh.out
```

## Translation - Behavior VHDL to Behavior Block Intermediate Format (bbif)

```
Command:

     vhdl2bbif [options]
<design_vhdl_filename>
                    -b  <filename>  :
Publish BBIF.
                    -bv <filename>  :
Publish BBIF in VHDL.
     If <filename> not provided,
standard output is used.
```

The next step is to translate the behavior description into an intermediate format suited for high-level synthesis. The vhdl2bbif translator takes the behavior vhdl description and produces an internal representation of the design. in the Behavior Block Intemediate Format (bbif) that can later be provided to the high-level synthesis tool (asserta). The vhdl2bbif translator can be invoked as follows:

```
vhdl2bbif -b design.bbif -bv
design_bbif.vhd design.vhd
```

- Input : design.vhd - vhdl file containing the behavioral specification of the design.
- Outputs :
  - design.bbif - BBIF (ascii file) containing the internal representation of the design.
  - design_bbif.vhd - BBIF (vhdl file) that containing the internal representation of the design.

The BBIF ascii file along with a Component Library file serve as input to the Synthesis Tool asserta. The BBIF vhdl file contains the same internal representation of the design, except in a simulateable form. With proper insertion of wait statements at appropriate places in the BBIF vhdl file, it can be simulated along with the original testbench that was written along with the behavior vhdl file.

The BBIF although being an intermediate representation, is in a high readable and understandable form. Designers may edit the file in order to make any meaningful last-minute changes before going through high-level synthesis. Also if the user wishes to specify newly designed RTL components (other than the ones that we have provided) then a knowledge of the correspondence between the bbif and component libary is necessary. We provide here a brief description of the Behavior Block Intermediate Format (bbif). For a more detailed description, interested readers may download a postscript version of the BBIF document.

The Behavior Block Intermediate Format has only one data type, which we call a carrier. A carrier is represented in the file as a tuple "(X 16)", denoting that the carrier X is 16 bits wide. The bbif file consists of the following sections:

- Name of the design specification, denoted by "(SPEC ....)".
- A list of carriers denoted by "(INPORT .....)" that represent the design input ports.
- A list of carriers denoted by "(OUTPORT .....)" that represent the design output ports.

• A collection of Behavior Blocks, each block beginning with a "(BB  Block_name.....".

Each Behavior Block (BB) can be viewed as a procedure in a conventional programming language. A BB consists of the following sections:

- The name of the block. For example "(BB Blk_5...."
- A list of carriers that a formal inputs to the block. For example "(BB Blk_5 ( (X 16) (Y 17) )", denotes that X and Y are formal inputs to Blk_5.
- A list of carriers that are locals to that block. For example, "(LOCAL (P 5) (Q 8))".
- A list of carriers that are constants. For example, "(CONSTANT (C 4 0101))". Constants have an extra field that denote that actual value of the constant. In this case, C represents the four-bit contant value "0101".
- A list of function statements, each statement of the form:
  stmt_number  (function_name  (input carrier names)  (output carrier names))

Each operation in the input VHDL file is converted into a corresponding bbif function inside a BB. For example, if the VHDL file contains an equation of the form:

```
a = b * c
```

It is translated into a bbif function of the form:

```
bb_mult (b c) (a)
```

NOTE : For each function name used in the bbif file, the component library must contain a component that can implement it. In other words, the function name must appear in the MODE field (see the following section that described the component library) of atleast one component in the component library.

**Illustrative Example : TLC**

```
Command used                      : vhdl2bbif
-bv tlc_bbif.vhd -b tlc.bbif tlc.vhd
Translated BBIF VHDL file    : tlc-bbif.vhd
Translated BBIF file         : tlc.bbif
```

# Simulation after translation(optional)

After the behavior VHDL file has been translated into the BBIF (acii and vhdl) files, the BBIF vhdl file must be simulated to ensure correct translation into bbif. With proper insertion of wait statements at appropriate places in the BBIF vhdl file, it can be simulated along with the original testbench that was written along with the behavior vhdl file. The translator modifies all data types and converts them to "STD_LOGIC_VECTOR" data type. So, in order to successfully simulate the design at this stage, the test bench has to be suitably modified so that any data type other than "STD_LOGIC_VECTOR" is converted into "STD_LOGIC_VECTOR". For example, if the original behavior VHDL specification had a port declaration " Ain : in bit ", the translator converts this into the following port declaration " Ain : in std_logic_vector(1 downto 1) ". Therefore the testbench has to be modified to reflect this change as you will be simulating the translated bbif VHDL file.

The file"bbif_library.vhd" must be analyzed before analyzing any other files. This file contains a collection of functions that are required to simulate the bbif vhdl file. Again the synopsys simulation tools can be used:

```
vhdlan <bbif_library.vhd>
<design_bbif.vhd> <testbench.vhd>
vhdlsim <configuration>
```

**Illustrative Example : TLC**

```
TestBench Specification       : tlc-tb.vhd
Commands used                 : vhdlan
bbif_library.vhd tlc_bbif.vhd tlc_tb.vhd
                                vhdlsim E
Output of Simulation          : beh.out
```

## Specifying the component library

For the purposes of synthesis, we have provided a standard component library "hls_components.lib" that can implement any of the basic operations specified in the behavioral VHDL subset described earlier. Each of these components also have a corresponding pre-synthesized counterpart that will be used by the synopsys logic synthesis tool. However, if the user wishes to use other component that he has pre-synthesized, then he can instruct the high-level synthesis tool recognize those compoents by including them in the component library.

In this section we will briefly describe the component library specification and its correspondence to the bbif file. The Component Library file consists of a list of component declarations. For each component declared in the component library there should also be a pre-synthesized synopsys component, that the synopsys tool can later use during logic synthesis. An example component declaration is given below:

```
(COMP multiplier (input_width  output_width)
   (CLASS ALU)
   (MODE bb_mult)
   (INPORT (1 input_width))
   (OUTPORT (1 output_width))
   (CONTROL )
)
```

The declaration denotes that there is a pre-synthesized component called "multiplier". Note that the mode field denotes that component is capable of implementing the bbif function "bb_mult". Therefore the BBIF file and Component Library file have a direct correspondence. Every function in the BBIF file (example "bb_mult") must have a corresponding component in the Component Library file that can implement it, i.e whose MODE field contains the function name.

If the user wishes tó provide newly designed components, or change the components specified in the library that we have provided, he may do so adhering to the syntax specified in the Component Library Document. Note that the component library must contain some pre-defined components essential for high-level synthesis.

The component library given as the Illustrative example may be used as the component library of any design that the user wishes to synthesize.

```
Illustrative Example : TLC
    Use the Standard Component Library
    Specification : hls components.lib
```

## High Level Synthesis using asserta

Once the bbif file is obtained and the component library has been created, then next step involves performing High Level Synthesis. The following command should be used:

```
asserta <bbif file> <component library file>
```

The Synthesis process prompts the user to select a specific Resource Set (collection of components to be synthesized into the RTL datapath). The following query appears on the screen:

```
Enter ranges for Resource Sets
(y/n) :
```

The user can either answer 'n' wherein the largest possible (fastest design) combination is synthesized. If the user answers 'y' and then he can explicitly specify the number of resources for each Resource that should be used. If the answer is 'y', then the user will be asked to enter the ranges for each Resource (the total number of resources of each resource type). For example,

```
Enter Count for Resource R1(1-4):
```

The query shows that for resource R1, the maximum possible instances is four and the minimum is one. The user can choose a count of three instances to be synthesized, by entering 3 as the answer.

The Synthesis process terminates with the following query

```
Enter name of the rtl file:
```

Enter a file name where the output of asserta has to be stored. A good practice is to provide the design_name here.

**Illustrative Example : TLC**

```
    Command: asserta tlc.bbif tlc.comp
             Enter ranges for Resource Sets
    (y/n) : n
             Enter name of rtl file : tlc

    Synthesized RTL Output: tlc.rtl
```

## Translation - asserta RTL to VHDL using rtl2vhdl

The output of the high-level synthesis tool asserta is also an intermediate representation. This representation has to be converted into the corresponding VHDL RTL representation. To achieve this purpose, execute the following command.

```
    rtl2vhdl <component library file>
<rtl file> <design name>
```

The component library file is the same file that was input to the high-level synthesis tool. This will produce 3 files containing the VHDL representations of the datapath, the controller and the overall design : design_name_dp.vhd  design_name_con.vhd design_name_des.vhd.

```
Illustrative example : TLC
    Command used              : rtl2vhdl
    tlc.comp tlc.rtl tlc
    Outputs of rtl2vhdl       : tlc dp.vhd
                               tlc con.vhd
                               tlc des.vhd
```

## Simulation at RTL

The next step is to simulate the design at the rtl vhdl level. In order to do this, modify your ".synopsys_vss.setup" file so that lib_fpga now points to the simulation library. That is, the following line should be uncommented

```
     lib_fpga :
/home/ddel/public_html/projects/asserta/rtl_components/
```

and the following line should be commented out

```
     lib_fpga :
/home/ddel/public_html/projects/asserta/rtl_components/
```

Once this has been done, the following commands are invoked:

```
    vhdlan <design_name_dp.vhd>
    <design_name_con.vhd>
    <design_name_des.vhd>
    <testbench2.vhd>
    vhdlsim <config_name>
```

Ensure that the files are analyzed in the same order as specified above.

```
Illustrative example : TLC
    Testbench Spec : tlc_tb.vhd tlc_configs.vhd
    Commands used  : vhdlan tlc_dp.vhd
    tlc_con.vhd tlc_des.vhd tlc_tb.vhd
                          tlc_configs.vhd
                     vhdlsim tlc_config
    Output of sim  : rtl.out
```

# Logic and Layout Synthesis using rtlvhdl2fpga

The resulting RTL design (consisting of the three vhdl files along with the pre-synthesized RTL component library) must now be taken through the Synopsys Logic Synthesis tool (fpga_compiler) targeted for the Xilinx 4000 family. The fpga_compiler will then produce an "xnf" file that must then be taken through Xilinx M1 Layout Synthesis tools to obtain a bitmap file for the fpga devide. We have provided a script "rtlvhdl2fpga" that invokes the Synopsy fpga_compiler followed by the Xilinx M1 tools, to synthesize an RT level VHDL specification, onto any Xilinx FPGA device.

Before running the "rtlvhdl2fpga" script, modify your ".synopsys_vss.setup" file so that lib_fpga now points to the synthesis library. That is, the following line should be uncommented

```
    lib_fpga :
/home/ddel/public_html/projects/asserta/rtl_components/
```

and the following line should be commented out

```
    lib_fpga :
/home/ddel/public_html/projects/asserta/rtl_components/
```

The "rtlvhdl2fpga" script usage is as follows:

```
  Usage: rtlvhdl2fpga <arguments>
        The following arguments must be
provided:
              <design_entity_name>
(eg. tlc_behavior)
              <device-speedgrade>
(eg. 4013e-3)
```

```
               <package>
(eg. hq240)
               <rtl_file_name>
(eg. tlc)
       Note: The following three files
should be present in the
       current directory:
         tlc_dp.vhd tlc_con.vhd
tlc_des.vhd
       Some useful outputs:
         xilinx/5_trace/tlc.twr  -
design performance data
         xilinx/6_bit/tlc.ncd   -
placed design, viewable using EPIC
         xilinx/6_bit/tlc.bit   -
design bit map file
         xilinx/7_ba/tlc.xnf    - back
annotated xnf file
```

For example, if your design name (entity name in the original behavior vhdl specfication) was "tlc_behavior" and the rtl file name that you provided for the high-level synthesis tool (asserta) was "tlc" and you want the implementation on a Xilinx 4013e chip, 240 pin package, with speed grade -3, then the following command should do it:

```
    rtlvhdl2fpga tlc_behavior 4013e-3
   hq240 tlc
```

This command will create two directories "synopsys" and "xilinx", wherein the respective logic/layout synthesis tools will be run. The resulting bitmap file in the directory "xilinx/6_bit" can then be downloaded on the corresponding FPGA device. Also a back annotated xnf file in the directory "xilinx/7_ba" is produced. This file can be converted to a "wir" file (using the tool xnf2wir), and the "wir" file can be simulated using the Viewlogic "powerview" tool set (viewsim).

APPENDIX C

## BEHAVIOURAL VHDL CODE

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;
use std.textio.all;

entity tb_dwt is
end tb_dwt;

architecture testbench of tb_dwt is

component dwt
port(clk,reset: in std_logic;
                datain: in real;
                dataout: out real;
                filter: in integer);
end component;
        signal clk: std_logic;
        signal reset: std_logic:='1';
        signal datain: real;
        signal dataout: real;
        signal filter: integer;
begin

UUT: dwt port map
(clk=>clk,reset=>reset,datain=>datain,dataout=>dataout,filter=>filter);

        --Set up 100ns clock cycle (arbitrary!)
        clock:process
        variable clktmp: std_logic:='0';

        begin
                clktmp:=not clktmp;
                clk<=clktmp;
                wait for 50 ns;
        end process;

        --Stimulus required by unit under test
        stimulus:process

        begin
        --Reset signal asserted to initialite unit
        reset<='1';

        --Then after 25ns (arbitrary)
        wait for 25 ns;

        --Reset taken low starts process
        reset<='0';

        --Filter type signal sent to unit
        filter<=0;


        --Data to unit is supplied by file not stimulus

        --while not(endfile(cfile)) loop
        --readline(cfile,inline);
        --read(inline,inputdata);
        --wait until clk'event and clk='1';
        --datain<=125.0;
        wait for 50 ns;
        --datain<=125.0;

        --end loop;
        --file_close(cfile);
        wait;
```

```
end process;

end testbench;

configuration build of tb_dwt is
for testbench
for uut: dwt

end for;
end for;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.global.all;
use std.textio.all;

entity dwt is
port(   clk: in std_logic;
        reset: in std_logic;
        datain: in real;
        dataout: out real;
        filter: in integer);
end dwt;

architecture behavior of dwt is
        signal h0: coef;
        signal h1: coef;
        signal siso: cold:=(others=>0);
begin

--8 Different anlysis filters supplied, correct one is chosen by filter signal
coeff:process(filter)
begin
case filter is

when 0 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000675;
h0(2)<=0.000000;h1(2)<=0.000002;
h0(3)<=0.000000;h1(3)<=-0.006684;
h0(4)<=0.000000;h1(4)<=0.001788;
h0(5)<=0.020401;h1(5)<=0.021802;
h0(6)<=0.000058;h1(6)<=0.009733;
h0(7)<=-0.111388;h1(7)<=-0.060571;
h0(8)<=0.054299;h1(8)<=-0.115519;
h0(9)<=0.536628;h1(9)<=0.440781;
h0(10)<=0.536628;h1(10)<=-0.440781;
h0(11)<=0.054299;h1(11)<=0.115519;
h0(12)<=-0.111388;h1(12)<=0.060571;
h0(13)<=0.000058;h1(13)<=-0.009733;
h0(14)<=0.020401;h1(14)<=-0.021802;
h0(15)<=0.000000;h1(15)<=-0.001788;
h0(16)<=0.000000;h1(16)<=0.006684;
h0(17)<=0.000000;h1(17)<=-0.000002;
h0(18)<=0.000000;h1(18)<=-0.000675;
h0(19)<=0.000000;h1(19)<=0.000000;

when 1 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=-0.005991;h1(4)<=0.010028;
h0(5)<=0.002658;h1(5)<=-0.004449;
h0(6)<=0.033433;h1(6)<=-0.076889;
h0(7)<=-0.023670;h1(7)<=0.048906;
h0(8)<=-0.048704;h1(8)<=0.316861;
h0(9)<=0.271012;h1(9)<=-0.588912;
h0(10)<=0.542524;h1(10)<=0.316861;
h0(11)<=0.271012;h1(11)<=0.048906;
h0(12)<=-0.048704;h1(12)<=-0.076889;
h0(13)<=-0.023670;h1(13)<=-0.004449;
h0(14)<=0.033433;h1(14)<=0.010028;
h0(15)<=0.002658;h1(15)<=-0.000000;
h0(16)<=-0.005991;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=-0.000000;
```

```
h0(18)<=0.000000;h1(18)<=-0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 2 =>
 h0(0)<=0.000000;h1(0)<=0.000000;
 h0(1)<=0.000000;h1(1)<=0.000000;
 h0(2)<=0.000000;h1(2)<=0.000000;
 h0(3)<=0.000000;h1(3)<=0.000000;
 h0(4)<=0.000000;h1(4)<=0.000000;
 h0(5)<=0.000000;h1(5)<=0.000000;
 h0(6)<=0.000000;h1(6)<=0.000000;
 h0(7)<=0.000000;h1(7)<=-0.062500;
 h0(8)<=0.000000;h1(8)<=-0.062500;
 h0(9)<=0.500000;h1(9)<=0.500000;
 h0(10)<=0.500000;h1(10)<=-0.500000;
 h0(11)<=0.000000;h1(11)<=0.062500;
 h0(12)<=0.000000;h1(12)<=0.062500;
 h0(13)<=0.000000;h1(13)<=0.000000;
 h0(14)<=0.000000;h1(14)<=0.000000;
 h0(15)<=0.000000;h1(15)<=0.000000;
 h0(16)<=0.000000;h1(16)<=0.000000;
 h0(17)<=0.000000;h1(17)<=0.000000;
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 3=>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=0.000000;h1(4)<=0.000000;
h0(5)<=0.000000;h1(5)<=0.000000;
h0(6)<=0.000000;h1(6)<=0.000000;
h0(7)<=0.000000;h1(7)<=0.000000;
h0(8)<=-0.125000;h1(8)<=0.250000;
h0(9)<=0.250000;h1(9)<=-0.500000;
h0(10)<=0.750000;h1(10)<=0.250000;
h0(11)<=0.250000;h1(11)<=0.000000;
h0(12)<=-0.125000;h1(12)<=0.000000;
h0(13)<=0.000000;h1(13)<=0.000000;
h0(14)<=0.000000;h1(14)<=0.000000;
h0(15)<=0.000000;h1(15)<=0.000000;
h0(16)<=0.000000;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=0.000000;
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 4 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=0.000000;h1(4)<=0.000000;
h0(5)<=0.000000;h1(5)<=0.013374;
h0(6)<=0.000000;h1(6)<=-0.004942;
h0(7)<=-0.091272;h1(7)<=-0.047544;
h0(8)<=0.033722;h1(8)<=-0.094320;
h0(9)<=0.557544;h1(9)<=0.434907;
h0(10)<=0.557544;h1(10)<=-0.434907;
h0(11)<=0.033722;h1(11)<=0.094320;
h0(12)<=-0.091272;h1(12)<=0.047544;
h0(13)<=0.000000;h1(13)<=0.004942;
h0(14)<=0.000000;h1(14)<=-0.013374;
h0(15)<=0.000000;h1(15)<=0.000000;
h0(16)<=0.000000;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=0.000000;
```

```
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 5 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=0.000000;h1(4)<=0.000000;
h0(5)<=0.000000;h1(5)<=0.000000;
h0(6)<=0.023437;h1(6)<=0.000000;
h0(7)<=-0.046875;h1(7)<=0.000000;
h0(8)<=-0.125000;h1(8)<=0.250000;
h0(9)<=0.296875;h1(9)<=-0.500000;
h0(10)<=0.703125;h1(10)<=0.250000;
h0(11)<=0.296875;h1(11)<=0.000000;
h0(12)<=-0.125000;h1(12)<=0.000000;
h0(13)<=-0.046875;h1(13)<=0.000000;
h0(14)<=0.023437;h1(14)<=0.000000;
h0(15)<=0.000000;h1(15)<=0.000000;
h0(16)<=0.000000;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=0.000000;
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 6 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=0.000000;h1(4)<=0.000000;
h0(5)<=0.000000;h1(5)<=0.000000;
h0(6)<=0.026748;h1(6)<=-0.045636;
h0(7)<=-0.016864;h1(7)<=0.028771;
h0(8)<=-0.078223;h1(8)<=0.295636;
h0(9)<=0.266864;h1(9)<=-0.557544;
h0(10)<=0.602949;h1(10)<=0.295636;
h0(11)<=0.266864;h1(11)<=0.028771;
h0(12)<=-0.078223;h1(12)<=-0.045636;
h0(13)<=-0.016864;h1(13)<=0.000000;
h0(14)<=0.026748;h1(14)<=0.000000;
h0(15)<=0.000000;h1(15)<=0.000000;
h0(16)<=0.000000;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=0.000000;
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;

when 7 =>
h0(0)<=0.000000;h1(0)<=0.000000;
h0(1)<=0.000000;h1(1)<=0.000000;
h0(2)<=0.000000;h1(2)<=0.000000;
h0(3)<=0.000000;h1(3)<=0.000000;
h0(4)<=0.000000;h1(4)<=0.000000;
h0(5)<=0.000000;h1(5)<=0.000000;
h0(6)<=0.000000;h1(6)<=0.000000;
h0(7)<=0.000000;h1(7)<=0.000000;
h0(8)<=0.000000;h1(8)<=0.000000;
h0(9)<=0.500000;h1(9)<=0.500000;
h0(10)<=0.500000;h1(10)<=-0.500000;
h0(11)<=0.000000;h1(11)<=0.000000;
h0(12)<=0.000000;h1(12)<=0.000000;
h0(13)<=0.000000;h1(13)<=0.000000;
h0(14)<=0.000000;h1(14)<=0.000000;
h0(15)<=0.000000;h1(15)<=0.000000;
h0(16)<=0.000000;h1(16)<=0.000000;
h0(17)<=0.000000;h1(17)<=0.000000;
```

```
h0(18)<=0.000000;h1(18)<=0.000000;
h0(19)<=0.000000;h1(19)<=0.000000;
when others =>
null;
end case;
end process;

dwt:process(clk,reset)

        --Variables for control, temp storage, etc.
        variable outline: line;
        variable b: integer:=0;
        variable o: integer:=0;
        variable a,in_r,in_w: integer;
        variable j,d_l_l: integer;
        variable d_l:integer:=1;
        variable k,z: integer;
        variable k1: integer;
        variable j1,ino,jno: integer;

        --Process finished signal (handy for large images!)
        variable flag_out:integer:=0;

        --Level of decomposition to be applied
        variable liv: integer;

        variable cont,contw: integer;
        variable livtemp: integer;
        variable flag,go_flag: integer;
        variable op: operation;

        --Temp storage for intermediate results
        variable nwx: new_x;
        variable xx: x:=(others=>0.0);
        variable yyl: yl;
        variable yyh: yh;

        --Image data is available at this file
        file input_file :text is in "tst_data_in";

        --Decomposed image is here, subbands not ordered in any way
        file output_file:text is out "tst_data_out";

        --Raster scan applied to subbands for multilevel type decomposition
        file out_file:text is out "tst_o_data_out";

        --Variables to do with file application issues
        variable inline: LINE;
        variable outl,outlo: LINE;
        variable inputdata: integer;
        variable temp,tempo: real;

        --Storage area
        variable ima:image_matrix;
        variable imao:image_o_matrix;

        --Control of storage area
        variable control: integer;
        variable clock: integer:=0;

begin

if reset ='1'then

        --Initialite unit
        contw:=0;
        j:=0;
        k:=0;
        j1:=1;
        k1:=1;
        cont:=11;
        op := frow;
        liv:=1;
        flag:=1;
        go_flag:=1;

                --Fill memory from data file
                if o=0 then
                        for i in 0 to (row*column-1) loop
                                readline(input_file,inline);
                                read(inline,inputdata);
                                temp:= real(inputdata);
                                ima(i):=temp;
                        end loop;
                        o:=0+1;
                end if;

elsif clk'event and clk='1' then

--Image decomposition control
if liv<=level and op=frow  then
                if cont /= 0 then
                   cont:=cont-1;
                   else
                   cont:=0;
                end if;

                if a<row then
                   if b<column then
                        k:=b;

                   else
                           k:=0;
                           j:=a;
                   end if;
                else
                        if b<column then
                                k:=b;
                        else
                                k:=0;
                                j:=0;
                                op:=frowt;
                                livtemp:=9;
                                k1:=k1*2;
                                flag:=0;
                        end if;
                end if;
elsif liv <=level and op=frowt then
        if livtemp>0 then
                livtemp:=livtemp-1;
        else
                op:=fcolumn;
                cont:=10;
                flag:=1;
                for i in 0 to 19 loop
                   xx(i):=0.0;
                end loop;
        end if;

elsif liv<=level and op=fcolumn then
        if cont /=0 then
                cont:=cont-1;
        else
                cont:=0;
        end if;
        if b<column then
                if a<row then
                        j:=a;
                else
                        j:=0;
                        k:=b;
                end if;
        else
                if a<row then
                        j:=a;
                else
                        j:=0;
                        k:=j1;
                        op:=fcolumnt;
                        livtemp:=9;
                        flag:=0;
                end if;
        end if;

elsif liv<=level and op=fcolumnt then
        if livtemp>0 then
                livtemp:=livtemp-1;
        else
                op:=fch;
                cont:=10;
                flag:=1;
                for i in 0 to 19 loop
                        xx(i):=0.0;
                end loop;
        end if;
elsif liv<=level and op=fch then
        if cont /=0 then
                cont:=cont-1;
        else
                cont:=0;      /
        end if;
        if b<column then
                if a<row then
                        j:=a;
                else
                        j:=0;
                        k:=b;
                end if;
        else
                if a<row then
                        j:=a;
                        else
                        k:=0;
                        j:=0;
                        op:=fcht;
                        livtemp:=9;
                        j1:=j1*2;
                        flag:=0;
                end if;
        end if;
elsif liv<=level and op=fcht then
        if livtemp>0 then
                livtemp:=livtemp-1;
        else
                op:=frow;
                cont:=10;
                flag:=1;
                for i in 0 to 19 loop
```

```
                        xx(i):=0.0;
                end loop;
                liv:=liv+1;
        end if;
else
--Write decomposed image (no ordering of subbands)
if contw=0 then
for i in 0 to (row*column-1) loop
write(outl,real(ima(i)));
writeline(output_file,outl);
end loop;


--Raster scan subbands process
for i in 1 to level loop

        for n in 0 to 0 loop
                d_l:=2*d_l;
        end loop;
        d_l_1:=d_l/2;

        for i in 0 to ((row/(d_l))-1) loop
                for j in (column/d_l) to ((column/(d_l_1))-1) loop
                        ino:=i*d_l;
                        jno:=(j-column/d_l)*d_l+d_l_1;
                        if row*column>(jno+column*ino) then
                                tempo:=ima(jno+column*ino);
                                imao(j+column*i):=tempo;
                        end if;
                end loop;
        end loop;

        for i in (row/d_l) to ((row/(d_l_1-0))-1) loop
                for j in (column/d_l) to ((column/(d_l_1))-1) loop
                        ino:=(i-row/d_l)*d_l+d_l_1;
                        jno:=(j-column/d_l)*d_l+d_l_1;
                        if row*column>(jno+column*ino) then
                        temp6:=ima(jno+column*ino);
                        imao(j+column*i):=tempo;
                        end if;
                end loop;
        end loop;

        for i in (row/d_l) to ((row/(d_l_1))-1) loop
                for j in 0 to ((column/(d_l))-1) loop
                        ino:=(i-row/d_l)*d_l+d_l_1;
                        jno:=j*d_l;
                        if row*column>(jno+column*ino) then
                        tempo:=ima(jno+column*ino);
                        imao(j+column*i):=tempo;
                        end if;
                end loop;
        end loop;

        if i=level then
                for i in 0 to ((row/(d_l))-1) loop
                        for j in 0 to ((column/(d_l-1))-1) loop
                                ino:=i*d_l;
                                jno:=j*d_l;
                                if row*column>(jno+column*ino) then
                                tempo:=ima(jno+column*ino);
                                imao(j+column*i):=tempo;
                                end if;
                        end loop;
                end loop;
        end if;
end loop;

--Write raster scan subband image
for i in 0 to (row*column-1) loop
write(outlo,real(imao(i)));
writeline(out_file,outlo);
end loop;

--Done
flag_out:=1;
contw:=contw+1;
end if;
end if;

--Memory address control
a:=j+j1;
b:=k+k1;
in_r:=k+column*j;

--Decimate every other value
for i in 9 downto 1 loop
siso(i)<=siso(i-1);
end loop;
siso(0)<=in_r;
in_w:=siso(9);

--nwx holds target element in image to be applied to transform
if flag=1 then
        nwx:=ima(in_r);
else
        nwx:=0.0;
end if;

--Decimate every other value
for i in 19 downto 1 loop
xx(i):=xx(i-1);
end loop;
xx(0):=nwx;                     /

--Target element is multiplied by filter coefficients
if go_flag=1 then
if clock mod 2=0 then
                yyl:=0.0;
                for n in 0 to 19 loop
                        yyl:=yyl+h0(n)*xx(n);
                end loop;

        if cont=0 then
                --Write low pass DWT coefficient
                ima(in_w):=yyl;
        end if;
else
        yyh:=0.0;
                for n in 0 to 19 loop
                        yyh:=yyh+h1(n)*xx(n+1);
                end loop;

        if cont=0 then
                --Write high pass DWT coefficient
                ima(in_w):=yyh;
        end if;
end if;
end if;
clock:=clock+1;

end if;
end process;
end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;
use std.textio.all;

entity tb_idwt is
end tb_idwt;

architecture testbench of tb_idwt is

component idwt
port(clk,reset: in std_logic;
                datain: in real;
                dataout: out real;
                filter: in integer);
end component;
        signal clk: std_logic;
        signal reset: std_logic:='1';
        signal datain: real;
        signal dataout: real;
        signal filter: integer;
begin

UUT: idwt port map
(clk=>clk,reset=>reset,datain=>datain,dataout=>dataout,filter=>filter);

       --Set up a 100ns clock cycle
        clock:process
        variable clktmp: std_logic:='0';

        begin
                clktmp:=not clktmp;
                clk<=clktmp;
                wait for 50 ns;
        end process;

        --Stimulus required by unit under test
        stimulus:process     /

        begin
        --reset pulse is sent to initialite unit
        reset<='1';

                --Then after 25 ns (arbitrary !)
                wait for 25 ns;

                --Reset is taken low so process starts
                reset<='0';

                --Filter type sent to unit
                filter<=0;

        --Data to unit is supplied from file

        --while not(endfile(cfile)) loop
        --readline(cfile,inline);
        --read(inline,inputdata);
        --wait until clk'event and clk='1';
        --datain<=125.0;
        wait for 50 ns;
        --datain<=125.0;

        --end loop;
        --file_close(cfile);
        wait;
end process;
end testbench;

configuration build2 of tb_idwt is
for testbench
for uut: idwt

end for;
end for;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.global.all;
use std.textio.all;

entity idwt is
port(   clk: in std_logic;
        reset: in std_logic;
        datain: in real;
        dataout: out real;
        filter: in integer);
end idwt;

architecture behavior of idwt is
        signal g0: coef;
        signal g1: coef;
        signal siso: cold:=(others=>0);
begin


coeff:process(filter)
begin

--7 Different filters supplied in filter storage
--filter signal is used to select right one
case filter is
when 0 =>
  g0(0)<=   0.000000;        g1(0)<=   0.000000;
  g0(1)<=   0.001349;        g1(1)<=   0.000000;
  g0(2)<=  -0.000004;        g1(2)<=   0.000000;
  g0(3)<=  -0.013367;        g1(3)<=   0.000000;
  g0(4)<=  -0.003375;        g1(4)<=   0.000000;
  g0(5)<=   0.043604;        g1(5)<=  -0.040803;
  g0(6)<=  -0.019467;        g1(6)<=   0.000116;
  g0(7)<=  -0.121143;        g1(7)<=   0.222775;
  g0(8)<=   0.231037;        g1(8)<=   0.108597;
  g0(9)<=   0.881563;        g1(9)<=  -1.073257;
  g0(10)<=  0.881563;        g1(10)<=  1.073257;
  g0(11)<=  0.231037;        g1(11)<= -0.108597;
  g0(12)<= -0.121143;        g1(12)<= -0.222775;
  g0(13)<= -0.019467;        g1(13)<= -0.000116;
  g0(14)<=  0.043604;        g1(14)<=  0.040803;
  g0(15)<= -0.003375;        g1(15)<=  0.000000;
  g0(16)<= -0.013367;        g1(16)<=  0.000000;
  g0(17)<= -0.000004;        g1(17)<=  0.000000;
  g0(18)<=  0.001349;        g1(18)<=  0.000000;
  g0(19)<=  0.000000;        g1(19)<=  0.000000;
when 1 =>
  g0(0)<=   0.000000;        g1(0)<=   0.000000;
  g0(1)<=   0.000000;        g1(1)<=   0.000000;
  g0(2)<=   0.000000;        g1(2)<=   0.000000;
  g0(3)<=   0.000000;        g1(3)<=   0.000000;
  g0(4)<=   0.000000;        g1(4)<=   0.000000;
  g0(5)<=   0.020056;        g1(5)<=   0.011983;
  g0(6)<=   0.008898;        g1(6)<=   0.005316;
  g0(7)<=  -0.153777;        g1(7)<=  -0.066867;
  g0(8)<=  -0.097811;        g1(8)<=  -0.047341;
  g0(9)<=   0.633722;        g1(9)<=   0.097408;
  g0(10)<=  1.177825;        g1(10)<=  0.542024;
  g0(11)<=  0.633722;        g1(11)<= -1.085048;
  g0(12)<= -0.097811;        g1(12)<=  0.542024;
  g0(13)<= -0.153777;        g1(13)<=  0.097408;
  g0(14)<=  0.008898;        g1(14)<= -0.047341;
  g0(15)<=  0.020056;        g1(15)<= -0.066867;
  g0(16)<=  0.000000;        g1(16)<=  0.005316;
```

```
   g0(17)<=  0.000000;        g1(17)<=  0.011983;
   g0(18)<=  0.000000;        g1(18)<=  0.000000;
   g0(19)<=  0.000000;        g1(19)<=  0.000000;
when 2 =>
   g0(0)<=   0.000000;        g1(0)<=   0.000000;
   g0(1)<=   0.000000;        g1(1)<=   0.000000;
   g0(2)<=   0.000000;        g1(2)<=   0.000000;
   g0(3)<=   0.000000;        g1(3)<=   0.000000;
   g0(4)<=   0.000000;        g1(4)<=   0.000000;
   g0(5)<=   0.000000;        g1(5)<=   0.000000;
   g0(6)<=   0.000000;        g1(6)<=   0.000000;
   g0(7)<=  -0.125000;        g1(7)<=   0.000000;
   g0(8)<=   0.125000;        g1(8)<=   0.000000;
   g0(9)<=   1.000000;        g1(9)<=  -1.000000;
   g0(10)<=  1.000000;        g1(10)<=  1.000000;
   g0(11)<=  0.125000;        g1(11)<=  0.000000;
   g0(12)<= -0.125000;        g1(12)<=  0.000000;
   g0(13)<=  0.000000;        g1(13)<=  0.000000;
   g0(14)<=  0.000000;        g1(14)<=  0.000000;
   g0(15)<=  0.000000;        g1(15) <=  0.000000;
   g0(16)<=  0.000000;        g1(16)<=  0.000000;
   g0(17)<=  0.000000;        g1(17)<=  0.000000;
   g0(18)<=  0.000000;        g1(18)<=  0.000000;
   g0(19)<=  0.000000;        g1(19)<=  0.000000;
when 3 =>
   g0(0)<=   0.000000;        g1(0)<=   0.000000;
   g0(1)<=   0.000000;        g1(1)<=  0.0000000;
   g0(2)<=   0.000000;        g1(2)<=  0.0000000;
   g0(3)<=   0.000000;        g1(3)<=   0.000000;
   g0(4)<=   0.000000;        g1(4)<=   0.000000;
   g0(5)<=   0.000000;        g1(5)<=   0.000000;
   g0(6)<=   0.000000;        g1(6)<=   0.000000;
   g0(7)<=   0.000000;        g1(7)<=   0.000000;
   g0(8)<=   0.000000;        g1(8)<=   0.000000;
   g0(9) <=   0.500000;       g1(9)<=   0.250000;
   g0(10)<=  1.000000;        g1(10)<=  0.500000;
   g0(11)<=  0.500000;        g1(11)<= -1.500000;
   g0(12)<=  0.000000;        g1(12)<=  0.500000;
   g0(13)<=  0.000000;        g1(13)<=  0.250000;
   g0(14)<=  0.000000;        g1(14)<=  0.000000;
   g0(15)<=  0.000000;        g1(15)<=  0.000000;
   g0(16)<=  0.000000;        g1(16)<=  0.000000;
   g0(17)<=  0.000000;        g1(17)<=  0.000000;
   g0(18)<=  0.000000;        g1(18)<=  0.000000;
   g0(19)<=  0.000000;        g1(19)<=  0.000000;
when 4 =>
   g0(0)<=   0.000000;        g1(0)<=   0.000000;
   g0(1)<=   0.000000;        g1(1)<=   0.000000;
   g0(2)<=   0.000000;        g1(2)<=   0.000000;
   g0(3)<=   0.000000;        g1(3)<=   0.000000;
   g0(4)<=   0.000000;        g1(4)<=   0.000000;
   g0(5)<=   0.026748;        g1(5)<=   0.000000;
   g0(6)<=   0.009884;        g1(6)<=   0.000000;
   g0(7)<=  -0.095087;        g1(7)<=   0.182544;
   g0(8)<=   0.188641;        g1(8)<=   0.067457;
   g0(9)<=   0.869813;        g1(9)<=  -1.115088;
   g0(10)<=  0.869813;        g1(10)<=  1.115088;
   g0(11)<=  0.188641;        g1(11)<=  0.067457;
   g0(12)<= -0.095087;        g1(12)<= -0.182544;
   g0(13)<=  0.009884;        g1(13)<=  0.000000;
   g0(14)<=  0.026748;        g1(14)<=  0.000000;
   g0(15)<=  0.000000;        g1(15)<=  0.000000;
   g0(16)<=  0.000000;        g1(16)<=  0.000000;
   g0(17)<=  0.000000;        g1(17)<=  0.000000;
   g0(18)<=  0.000000;        g1(18)<=  0.000000;
   g0(19)<=  0.000000;        g1(19)<=  0.000000;
```

```
when 5 =>
  g0(0)<=   0.000000;       g1(0)<=   0.000000;
  g0(1)<=   0.000000;       g1(1)<=   0.000000;
  g0(2)<=   0.000000;       g1(2)<=   0.000000;
  g0(3)<=   0.000000;       g1(3)<=   0.000000;
  g0(4)<=   0.000000;       g1(4)<=   0.000000;
  g0(5)<=   0.000000;       g1(5)<=   0.000000;
  g0(6)<=   0.000000;       g1(6)<=   0.000000;
  g0(7)<=   0.000000;       g1(7)<=  -0.046874;
  g0(8)<=   0.000000;       g1(8)<=  -0.093750;
  g0(9)<=   0.500000;       g1(9)<=   0.250000;
  g0(10)<=  1.000000;       g1(10)<=  0.593750;
  g0(11)<=  0.500000;       g1(11)<= -1.406250;
  g0(12)<=  0.000000;       g1(12)<=  0.593750;
  g0(13)<=  0.000000;       g1(13)<=  0.250000;
  g0(14)<=  0.000000;       g1(14)<= -0.093750;
  g0(15)<=  0.000000;       g1(15)<= -0.046874;
  g0(16)<=  0.000000;       g1(16)<=  0.000000;
  g0(17)<=  0.000000;       g1(17)<=  0.000000;
  g0(18)<=  0.000000;       g1(18)<=  0.000000;
  g0(19)<=  0.000000;       g1(19)<=  0.000000;
when 6 =>
  g0(0)<=   0.000000;       g1(0)<=   0.000000;
  g0(1)<=   0.000000;       g1(1)<=   0.000000;
  g0(2)<=   0.000000;       g1(2)<=   0.000000;
  g0(3)<=   0.000000;       g1(3)<=   0.000000;
  g0(4)<=   0.000000;       g1(4)<=   0.000000;
  g0(5)<=   0.000000;       g1(5)<=   0.000000;
  g0(6)<=   0.000000;       g1(6)<=   0.000000;
  g0(7)<=  -0.091272;       g1(7)<=  -0.053497;
  g0(8)<=  -0.057543;       g1(8)<=  -0.033728;
  g0(9)<=   0.591271;       g1(9)<=   0.156446;
  g0(10)<=  1.115088;       g1(10)<=  0.533727;
  g0(11)<=  0.591271;       g1(11)<= -1.205898;
  g0(12)<= -0.057543;       g1(12)<=  0.533727;
  g0(13)<= -0.091272;       g1(13)<=  0.156446;
  g0(14)<=  0.000000;       g1(14)<= -0.033728;
  g0(15)<=  0.000000;       g1(15)<= -0.053497;
  g0(16)<=  0.000000;       g1(16)<=  0.000000;
  g0(17)<=  0.000000;       g1(17)<=  0.000000;
  g0(18)<=  0.000000;       g1(18)<=  0.000000;
  g0(19)<=  0.000000;       g1(19)<=  0.000000;
when 7 =>
  g0(0)<=0.000000;          g1(0)<=0.000000;
  g0(1)<=   0.000000;       g1(1)<=   0.000000;
  g0(2)<=   0.000000;       g1(2)<=   0.000000;
  g0(3)<=   0.000000;       g1(3)<=   0.000000;
  g0(4)<=   0.000000;       g1(4)<=   0.000000;
  g0(5)<=   0.000000;       g1(5)<=   0.000000;
  g0(6)<=   0.000000;       g1(6)<=   0.000000;
  g0(7)<=   0.000000;       g1(7)<=   0.000000;
  g0(8)<=   0.000000;       g1(8)<=   0.000000;
  g0(9)<=   1.000000;       g1(9)<=  -1.000000;
  g0(10)<=  1.000000;       g1(10)<=  1.000000;
  g0(11)<=  0.000000;       g1(11)<=  0.000000;
  g0(12)<=  0.000000;       g1(12)<=  0.000000;
  g0(13)<=  0.000000;       g1(13)<=  0.000000;
  g0(14)<=  0.000000;       g1(14)<=  0.000000;
  g0(15)<=  0.000000;       g1(15)<=  0.000000;
  g0(16)<=  0.000000;       g1(16)<=  0.000000;
  g0(17)<=  0.000000;       g1(17)<=  0.000000;
  g0(18)<=  0.000000;       g1(18)<=  0.000000;
  g0(19)<=  0.000000;       g1(19)<=  0.000000;
when others =>
null;
end case;
end process;

dwt:process(clk,reset)

   --Variables for control, temp storage, etc.
       variable outline: line;
       variable b: integer:=0;
       variable o: integer:=0;
       variable a,in_r,in_w: integer;
       variable j,d_l_1: integer;
       variable d_l:integer:=1;
       variable k,z: integer;
       variable k1: integer;
       variable j1,ino,jno: integer;

       --This variable inicates process is finished (Very handy for large images)
       variable flag_out:integer:=0;

       variable liv: integer;
       variable cont,contw: integer;
       variable livtemp: integer;
       variable flag,go_flag: integer;
       variable op: operation;
       variable nwy: new_x;
       variable x_x:real;
       variable y_l: x:=(others=>0.0);
       variable y_h: x:=(others=>0.0);
       variable yyl: yl;
       variable yyh: yh;

       --Decomposed image data is tored in this file!
       file input_file :text is in "tst_data_out";

       --Transformed image will be stored in this file!
       file output_file:text is out "idwt_data_out";

       -- This variables have to do with file application issues
       variable inline: LINE;
       variable outl,outlo: LINE;
       variable inputdata: real;
       variable temp,tempo: real;
       variable ima:image_matrix;
       variable control: integer;

      --Important control variables
      variable clock,f_rit,cco: integer:=0;

begin

if reset ='1'then
        --Initialite all
        contw:=0;
        j:=0;
        a:=0;
        b:=0;
        k:=0;
        j1:=4;
        k1:=8;
        cont:=12;
        op := fcolumn;
        liv:=level;
        flag:=1;
        go_flag:=1;

                if o=0 then
                        --Fill memory from file
                        for i in 0 to (row*column-1) loop
                                readline(input_file,inline);
                                read(inline,inputdata);
                                temp:= real(inputdata);
                                ima(i):=temp;
                        end loop;
                        o:=0+1;
                end if;

elsif clk'event and clk='1' then
        if cco=0 then
                --Appropiate count control depends on filter chosen
                if filter=0 or filter=2 or filter=4 then
                        f_rit:=1;
                        cont:=11;
                        cco:=cco+1;
                end if;
        end if;

--Decomposed image data reconstruction control
if liv>0 and op=fcolumn  then
                        if cont /= 0 then
                           cont:=cont-1;
                        else
                           cont:=0;
                        end if;

                if b<column then
                   if a<row then
                        j:=a;

                   else
                           j:=0;
                           k:=b;
                   end if;
                else
                        if a<row then
                           j:=a;
                        else
                           j:=0;
                           k:=j1;
                           op:=fcolumnt;
                           livtemp:=9;
                           flag:=0;
                        end if;
                end if;
elsif liv>0  and op=fcolumnt then
        if livtemp>0 then
                livtemp:=livtemp-1;
        else
                op:=fch;
                cont:=10;
                flag:=1;
                clock:=clock+1;
                if f_rit=1 then
                        clock:=clock+1;
                end if;
                for i in 0 to 19 loop
                   y_l(i):=0.0;
                   y_h(i):=0.0;
                end loop;
         end if;

elsif liv>0 and op=fch then
        if cont /=0 then
                cont:=cont-1;
        else
                cont:=0;
        end if;
        if b<column then
                if a<row then
                        j:=a;
                else
                        j:=0;
                        k:=b;
                end if;
        else
                if a<row then
                        j:=a;
                else
                        j:=0;
                        k:=0;
                        op:=fcht;
                        livtemp:=9;
                        k1:=k1/2;
                        flag:=0;
                end if;
        end if;

elsif liv>0 and op=fcht then
        if livtemp>0 then
                livtemp:=livtemp-1;
        else
                op:=frow;
                cont:=10;
                flag:=1;
                clock:=clock+1;
                if f_rit=1 then
                        clock:=clock+1;
                end if;
                for i in 0 to 19 loop
                        y_l(i):=0.0;
                        y_h(i):=0.0;
                end loop;
        end if;
elsif liv>0 and op=frow then
        if cont /=0 then
                cont:=cont-1;
        else
                cont:=0;
        end if;
        if a<row then
                if b<column then
                        k:=b;
                else
                        k:=0;
                        j:=a;
                end if;
else
                if b<column then
                        k:=b;
                        else
                        k:=0;
                        j:=0;
                        op:=frowt;
                        livtemp:=9;
                        j1:=j1/2;
                        flag:=0;
                end if;
end if;
elsif liv>0 and op=frowt then
        if livtemp>0 then
```

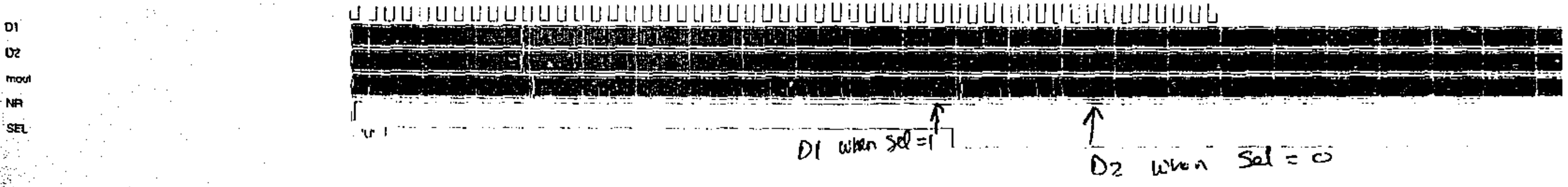
```
                livtemp:=livtemp-1;
        else
                op:=fcolumn;
                cont:=10;
                flag:=1;
                clock:=clock+1;
                if f_rit=1 then
                        clock:=clock+1;
                end if;
                liv:=liv-1;
                for i in 0 to 19 loop
                        y_l(i):=0.0;
                        y_h(i):=0.0;
                end loop;

        end if;
else

--When all is done write results in output file
if contw=0 then
        for i in 0 to (row*column-1) loop
                write(outl,real(ima(i)));
                writeline(output_file,outl);
        end loop;
        -- Done flag is active
        flag_out:=1;
        contw:=contw+1;
end if;
end if;

--Memory address control
a:=j+j1;
b:=k+k1;
in_r:=k+column*j;

--Adds zeros between decomposed image samples
for i in 12 downto 1 loop
    siso(i)<=siso(i-1);
end loop;
siso(0)<=in_r;
in_w:=siso(9);

--nwy holds decomposed image element according to in_r
if flag=1 then
        nwy:=ima(in_r);
else
        nwy:=0.0;
end if;

--Write the low pass resuts
if clock mod 2=0 then
        for i in 19 downto 1 loop
                y_l(i):=y_l(i-1);
        end loop;
        y_l(0):=nwy;
else
--Or high pass results
        for i in 19 downto 1 loop
                y_h(i):=y_h(i-1);

        end loop;
        y_h(0):=nwy;
end if;

-- Low/High pass coefficients  produced here (will produced recomposed data)
if go_flag=1 then
```

APPENDIX C

## STRUCTURAL VHDL CODE

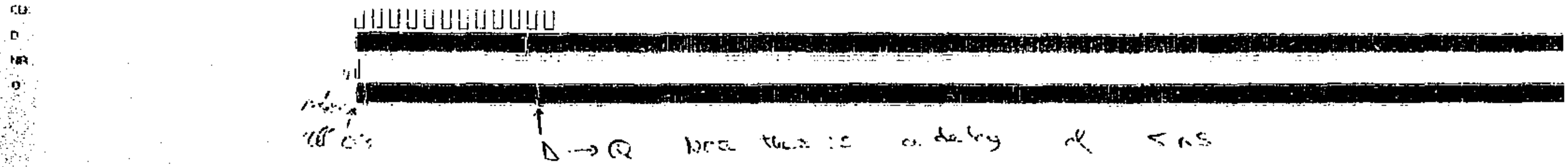
```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

--Design of a 2 channel multiplexer

entity MUX is

   Port ( CLK,SEL,NR: in std_logic;
          D1: in std_logic_vector (20 downto 0);
          D2: in std_logic_vector (20 downto 0);
          Mout: out std_logic_vector (20 downto 0));

end MUX;

Architecture Sequential of MUX is

begin
   Process (CLK, SEL)
   begin
   IF (NR = '0') then
   -- reset output
   Mout<=(others=>'0');
   ELSIF (CLK'event and CLK -'1') then
      if (SEL='0') then
         Mout<= D1;
      end if;
      if (SEL='1') then
         Mout<= D2;
      end if;
   END IF;
    End process;
end sequential;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

-- Test bench to test MUX

entity TB_MUX is
end TB_MUX;

architecture MUX_ARCH of TB_MUX is

      Signal CLK: std_logic:='0';
      Signal NR,SEL: std_logic;
      Signal D1,D2: std_logic_vector (20 downto 0);
      Signal mout: std_logic_vector (20 downto 0);


Component MUX
  port ( CLK,SEL, NR: in std_logic;
         D1: in std_logic_vector(20 downto 0);
          D2: in std_logic_vector(20 downto 0);
          Mout: out std_logic_vector(20 downto 0));

end Component;

begin

    UUT: MUX port map (CLK,SEL,NR,D1,D2,Mout);

   D1<="011111111000000000000" after 100 ns;-- 255.0000 in
      D2<="011111111000011111111" after 100 ns; -- 255.0000 in
   --Data<="1111111110000" after 250 ns; -- neg 255 in
   --coeff<="0010100000000" after 250 ns; -- 0.5 coeff
   --Data<="011111111000000000000" after 370 ns; -- 255 in
   --Coeff<="1010100000000" after 420 ns; -- neg 0.5 coeff

   NR <= '0' after 0 ns,
         '1' after 2 ns;
        -- '0' after 150 ns,
         --'1' after 200 ns,
         --'0' after 350 ns,
         --'1' after 400 ns;

   SEL<= '1' after 20 ns,
         '0' after 350 ns;
   CLK <= not CLK after 5 ns;

end MUX_ARCH;
```


D1
D2
mout
NR
SEL
D1 when Sel = 1
D2 when Sel = 0

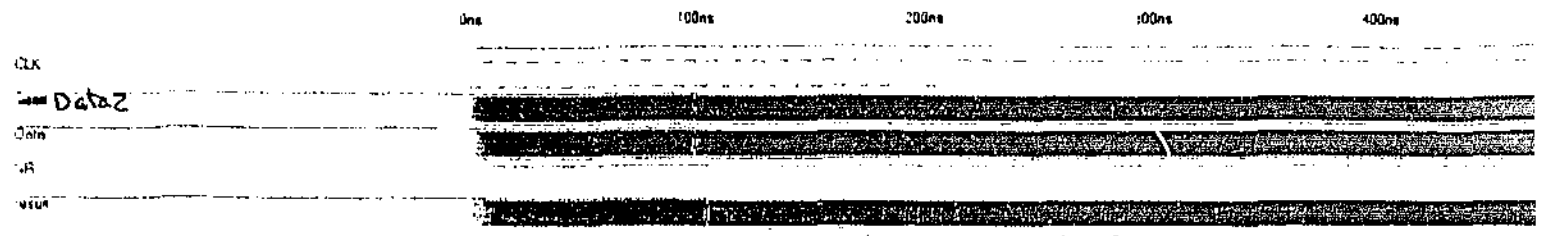
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--Design of a d type flip flop cell structure

entity DFF is

  port( CLK,NR:in std_logic;
  D: in std_logic_vector (20 downto 1);
  Q: out std_logic_vector (20 downto 1));

end DFF;

-- To get structure several of these cells can be joined
-- togerther

Architecture Sequential of DFF is
begin
   process (CLK,NR)
   begin
      if (NR = '0') then
      -- reset output to prevent errors
        Q<=(others=> '0');

      --on clock event shift data out
      elsif (CLK'event and CLK ='1') then

         Q <= D;

       end if;
     end process;
end Sequential;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- Test bench to test DFF

entity TB_DFF is
end TB_DFF;

architecture DFF_ARCH of TB_DFF is

    Signal CLK: std_logic:='0';
    Signal NR: std_logic;
    Signal D: std_logic_vector (20 downto 1);
    Signal Q: std_logic_vector (20 downto 1);

Component DFF
  port ( CLK, NR: in std_logic;
         D: in std_logic_vector(20 downto 1);
         Q: out std_logic_vector(20 downto 1));
end component;

begin

    UUT: DFF port map (CLK,NR, D, Q);

   D<="00000000000000000011" after 100 ns;

   NR <= '0' after 0 ns,
         '1' after 2 ns;


   CLK <= not CLK after 5 ns;

end DFF_ARCH;
```


CLK
D
NR
Q
all 0's
D → Q here there is a delay of < nS

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

--Design of a generic signed ~~multiplier~~ Adder cell structure

entity ADD is
  Generic (N: integer:=16);
  port( CLK,NR:in std_logic;
        Data:  in std_logic_vector (12 downto 0);
        Coeff: in std_logic_vector (12 downto 0);
        Result:out std_logic_vector(12 downto 0));


end ADD;



-- To get structure several of these cells can be joined
-- together

Architecture Sequential of ADD is
begin
 process (CLK,NR)

   variable mult_result: signed(12 downto 0):=(others=>'0');
 begin
    if (NR = '0') then

      --RESET ALL
      Result <= (others=>'0');
      --on clock event multiply data by high pass coeffient
    elsif (CLK'event and CLK ='1') then


         result<=unsigned(coeff)+unsigned(data);
    end if;
 end process;
end Sequential;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

-- Test bench to test ~~MULT~~ADD

entity TB_ADD is
end TB_ADD;

architecture ADD_ARCH of TB_ADD is

      Signal CLK: std_logic:='0';
      Signal NR: std_logic;
      Signal Data: std_logic_vector (12 downto 0);
      Signal Coeff: std_logic_vector (12 downto 0);
      Signal Result: std_logic_vector (12 downto 0);

Component ADD
  port ( CLK, NR: in std_logic;
         Data: in std_logic_vector(12 downto 0);
         coeff: out std_logic_vector(12 downto 0);
         Result: out std_logic_vector (12 downto 0));
end component;

begin

    UUT: ADD port map (CLK,NR, Data,Coeff,result);

   Data<="0111111110000" after 100 ns;-- 255.0000 in
   Coeff<="0111111110000" after 100 ns; -- 255.0000 in
   --Data<="1111111110000" after 250 ns; -- neg 255 in
   --coeff<="0010100000000" after 250 ns; -- 0.5 coeff
   --Data<="0111111110000000000" after 370 ns; -- 255 in
   --Coeff<="1010100000000" after 420 ns; -- neg 0.5 coeff

   NR <= '0' after 0 ns,
         '1' after 2 ns;
       -- '0' after 150 ns,
        --'1' after 200 ns,
        --'0' after 350 ns,
        --'1' after 400 ns;


   CLK <= not CLK after 5 ns;

end ADD_ARCH;
```

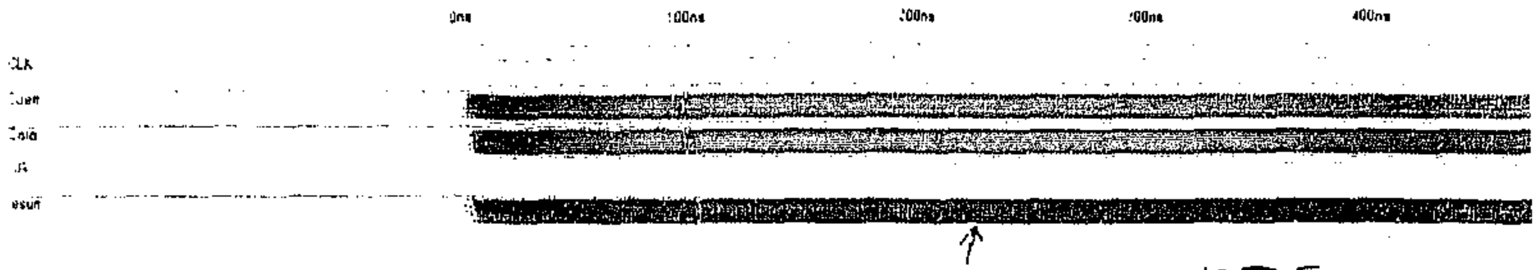


Datain = 255+255

Result = 510 = 111111110000

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

-- Test bench to test MULT

entity TB_MULT is
end TB_MULT;

architecture MULT_ARCH of TB_MULT is

      Signal CLK: std_logic:='0';
      Signal NR: std_logic;
      Signal Data: std_logic_vector (12 downto 0);
      Signal Coeff: std_logic_vector (12 downto 0);
      Signal Result: std_logic_vector (25 downto 0);

Component Mult
  port ( CLK, NR: in std_logic;
         Data: in std_logic_vector(12 downto 0);
         coeff: out std_logic_vector(12 downto 0);
          Result: out std_logic_vector (25 downto 0));
end component;

begin

    UUT: MULT port map (CLK,NR, Data,Coeff,result);

   Data<="0111111110000" after 100 ns;-- 255.0000 in
   Coeff<="0010100000000" after 100 ns; -- 0.5 coeff
   --Data<="1111111110000" after 250 ns; -- neg 255 in
   --coeff<="0010100000000" after 250 ns; -- 0.5 coeff
   --Data<="01111111100000000000" after 370 ns; -- 255 in
   --Coeff<="1010100000000" after 420 ns; -- neg 0.5 coeff

   NR <= '0' after 0 ns,
         '1' after 2 ns;
        -- '0' after 150 ns,
         --'1' after 200 ns,
         --'0' after 350 ns,
         --'1' after 400 ns;


   CLK <= not CLK after 5 ns;

end MULT_ARCH;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

--Design of a generic signed multiplier cell structure

entity MULT is
  Generic (N: integer:=16);
  port( CLK,NR:in std_logic;
        Data:  in std_logic_vector (12 downto 0);
        Coeff: in std_logic_vector (12 downto 0);
        Result:out std_logic_vector(25 downto 0));


end MULT;

--PRECISION OF DATA IS (DETERMINED FROM EXPERIMENTS IN FIRST
--BEHAVIOURAL MODEL) DATA IN IS 19 BIT TWOS COMPLEMENT, 1 BIT
--SIGN, 8 BIT INTEGER PART AND 10 BIT FRACTION PART.
--FILTER COEFFICIENT IS AN 18 BIT TWOS COMPLEMENT NUMBER,
--1 BIT SIGN, 1 BIT INTEGER, 16 BIT FRACTION PART.
--RESULT IS A 20 BIT TWOS COMPLEMENT NUMBER, 1 BIT SIGN,
--8 BIT INTEGER,11 BIT FRACTION PART.

-- To get structure several of these cells can be joined
-- together

Architecture Sequential of MULT is
begin
 process (CLK,NR)

   variable mult_result: signed(12 downto 0):=(others=>'0');
 begin
    if (NR = '0') then

       --RESET ALL
       Result <= (others=>'0');
       --on clock event multiply data by high pass coeffient
    elsif (CLK'event and CLK ='1') then


         result<=unsigned(coeff)*unsigned(data);
    end if;
 end process;
end Sequential;
```



255 x 0.5 = 12~~7~~5

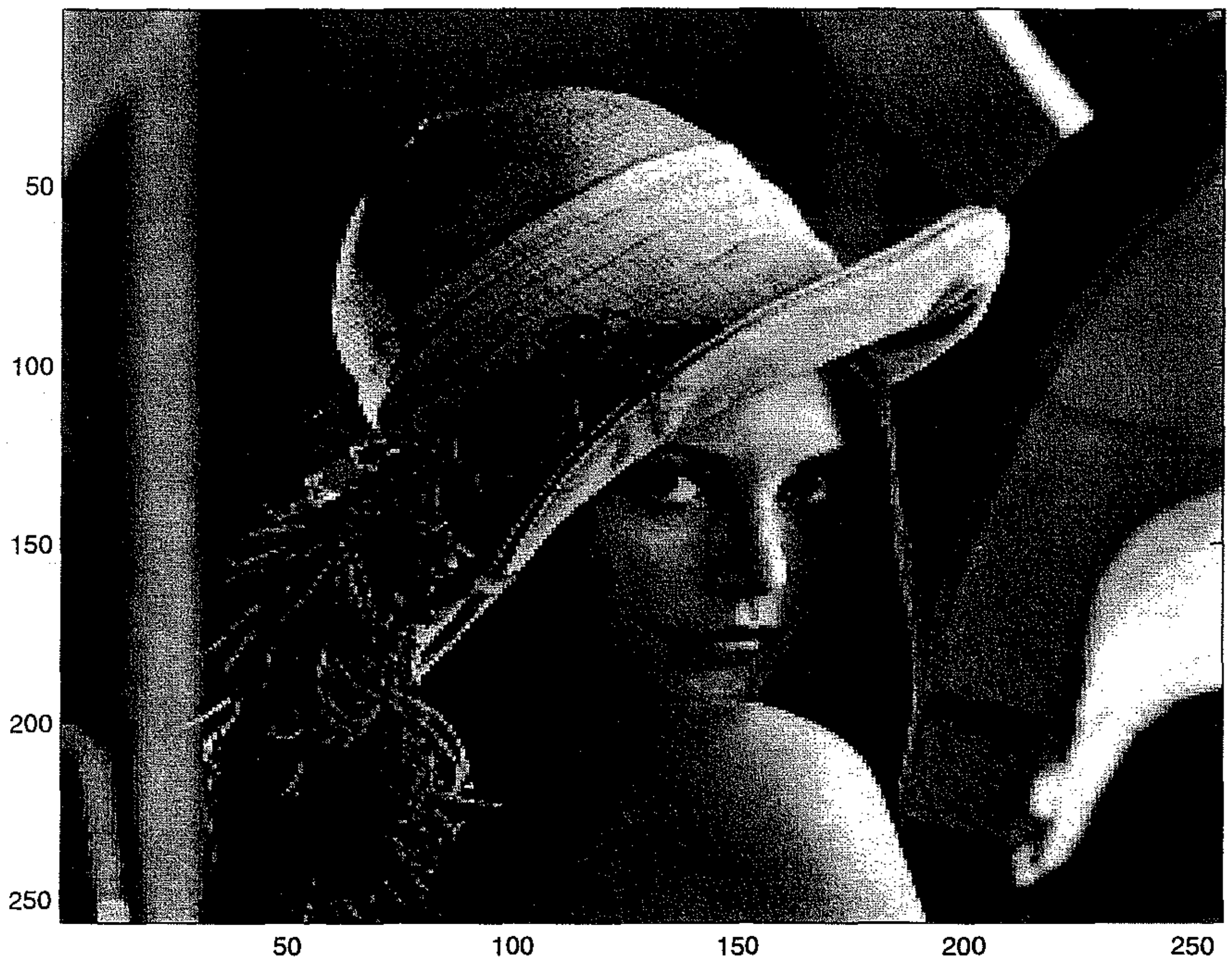
Result = 127 ( Rounded off

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

--Test bench to test RAM

entity TB_RAM is
end TB_RAM;

architecture RAM_ARCH of TB_RAM is

   Signal AE: std_logic;
   Signal RW: std_logic;
   Signal address: std_logic_vector(20 downto 0);
   Signal data: std_logic_vector(20 downto 0);

component RAM

   Port( AS,RW: in std_logic;
         address: in std_logic_vector(20 downto 0);
         data: inout std_logic_vector(20 downto 0));
end component;

begin

   UUT: RAM port map (AE,RW,ADDRESS,DATA);

   AE<='0' after 50 ns;
   AE<='1' after 100 ns;
   ADDRESS<="000000000000000000001" after 150ns;
   RW<='0' after 100 ns;
   DATA<="000000000000000000101" after 120ns;
   AE<='0' after 200 ns;
   AE<='1' after 250 ns;
   RW<='1' after 300 ns;
   ADDRESS<="000000000000000000001" after 280ns;

end RAM_ARCH;
```

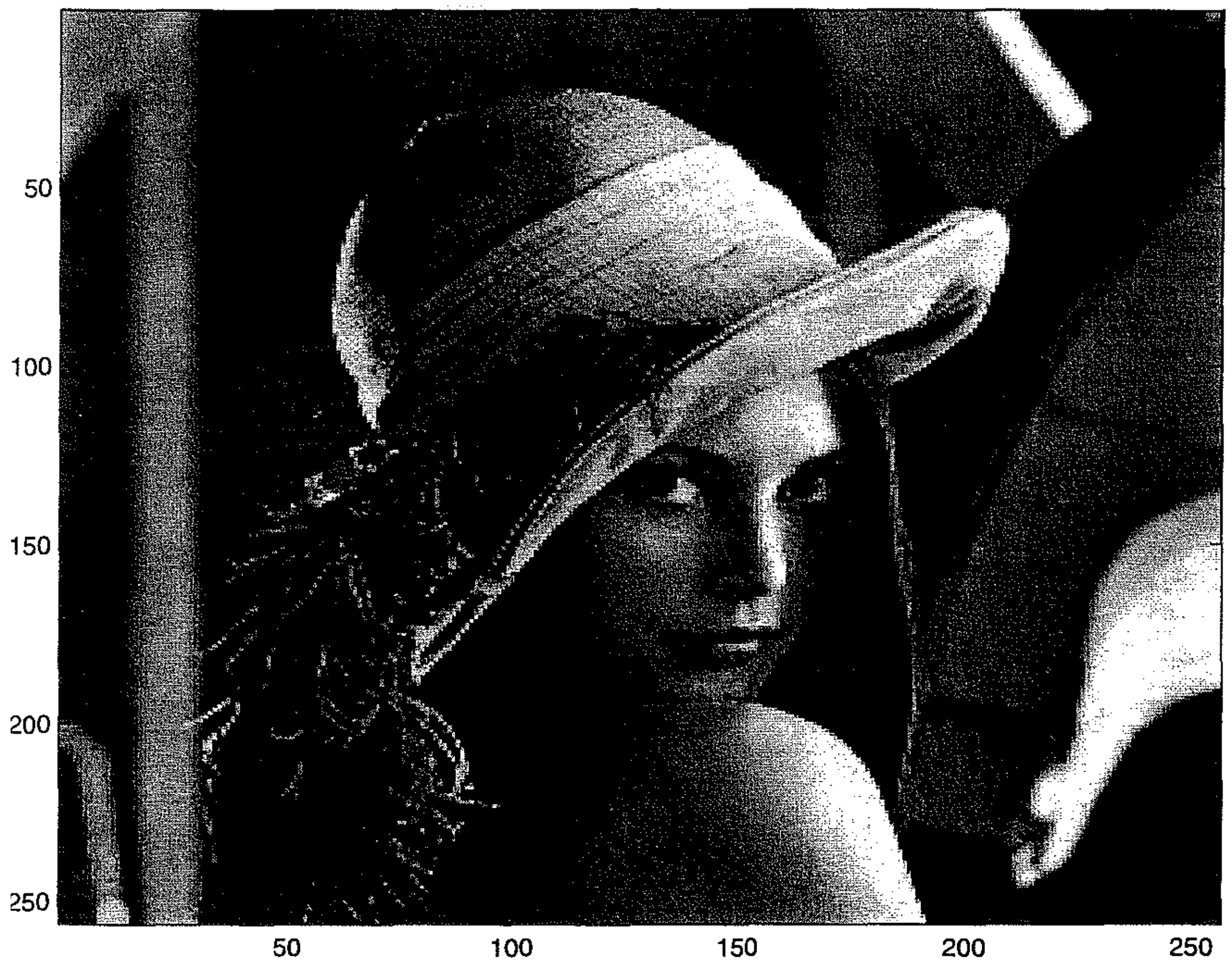
```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Design of a RAM cell

Entity RAM is
   Generic(n: integer:=256);
   Port(AE,RW: in std_logic;
        Address: in std_logic_vector(20 downto 0);
             data: inout std_logic_vector(20 downto 0));

end RAM;

Architecture Sequential of RAM is

type rmbit is array (0 to n) of std_logic_vector(20 downto 0);

begin

   Process (AE,RW,Address,data)
     variable dram: rmbit;
     begin
        if (AE ='0') then
           data <= "000000000000000000000";
        elsif (AE'event and AE='1') then
           if (RW = '1') then
             data<=dram(address);
           end if;
           if (RW = '0') then
            dram(address):=data;
           end if;
        end if;

    end process;
end sequential;
```

APPENDIX C

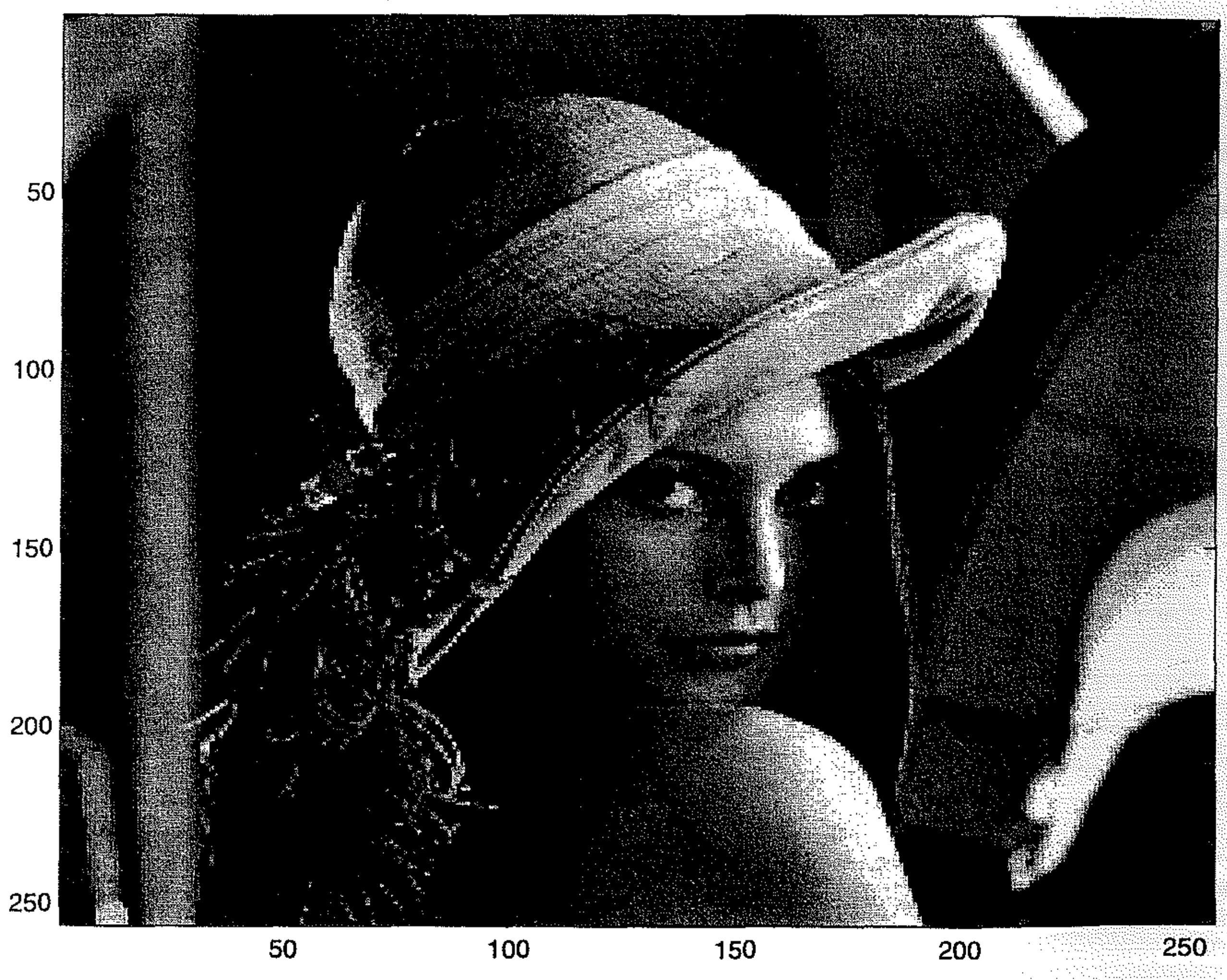
## RESULTS



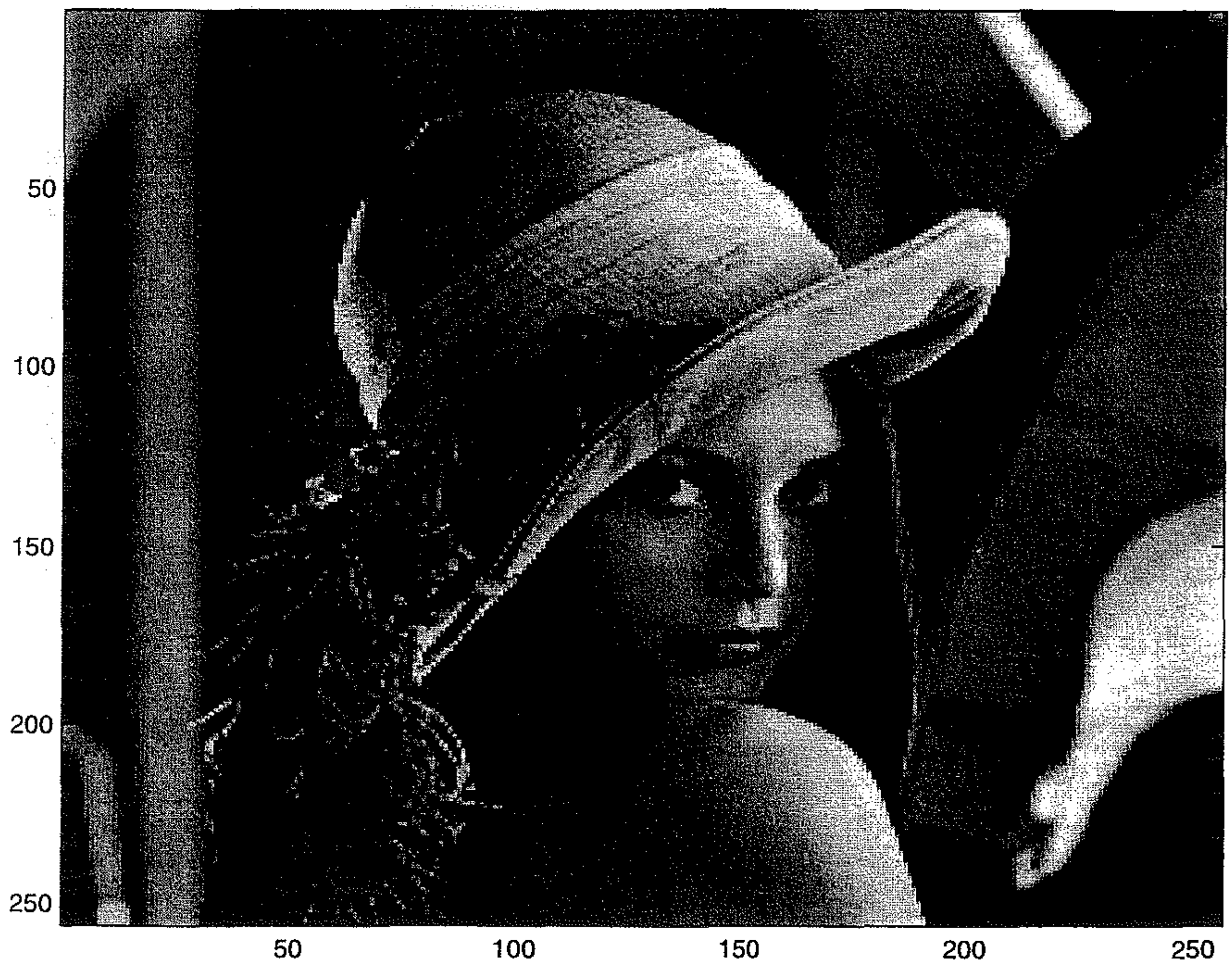
Filter Type 2-2
Level 3 inverse transformed image
MSE = 0



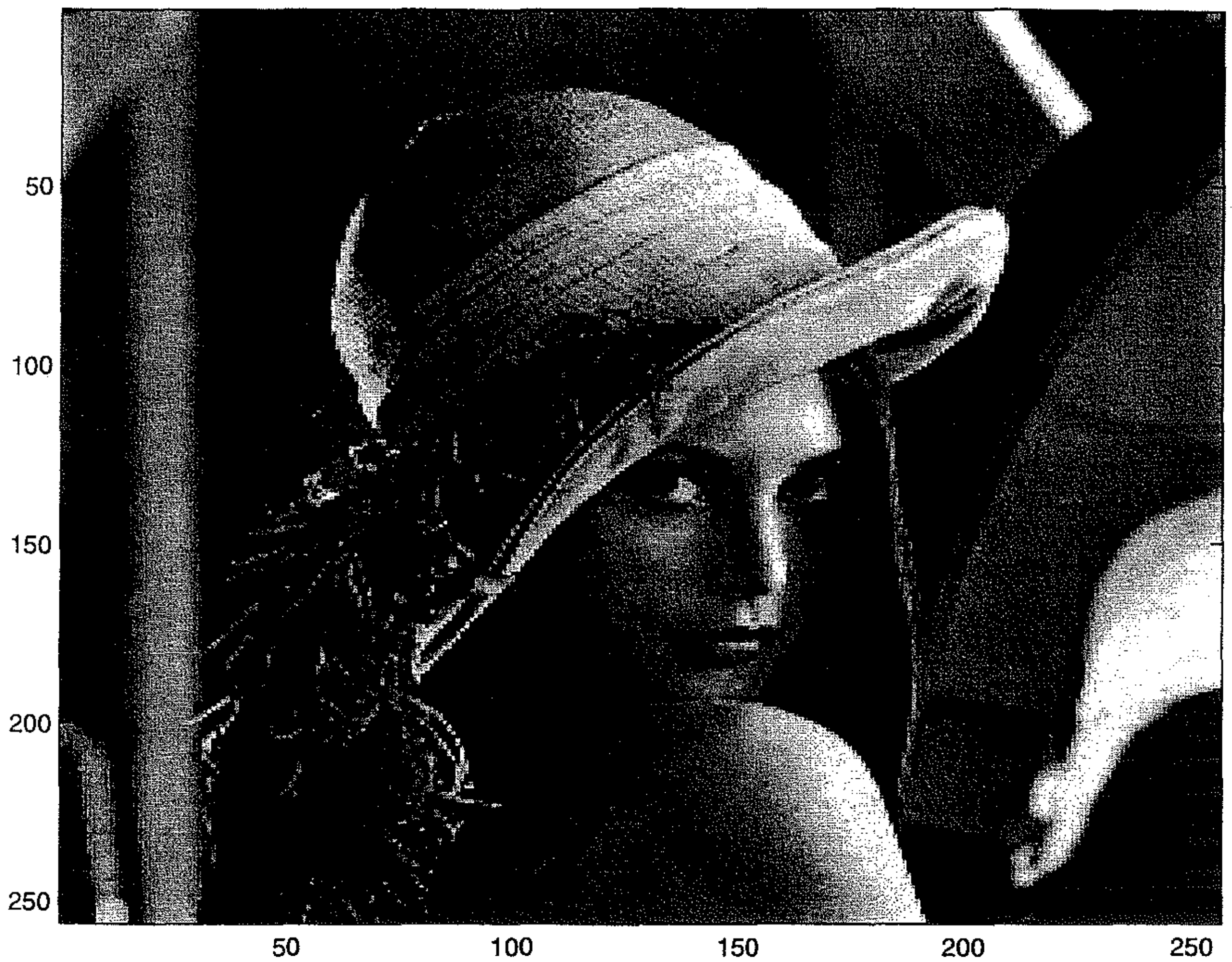
Filter Type 2-6
Level 3 inverse transformed image
MSE = 0



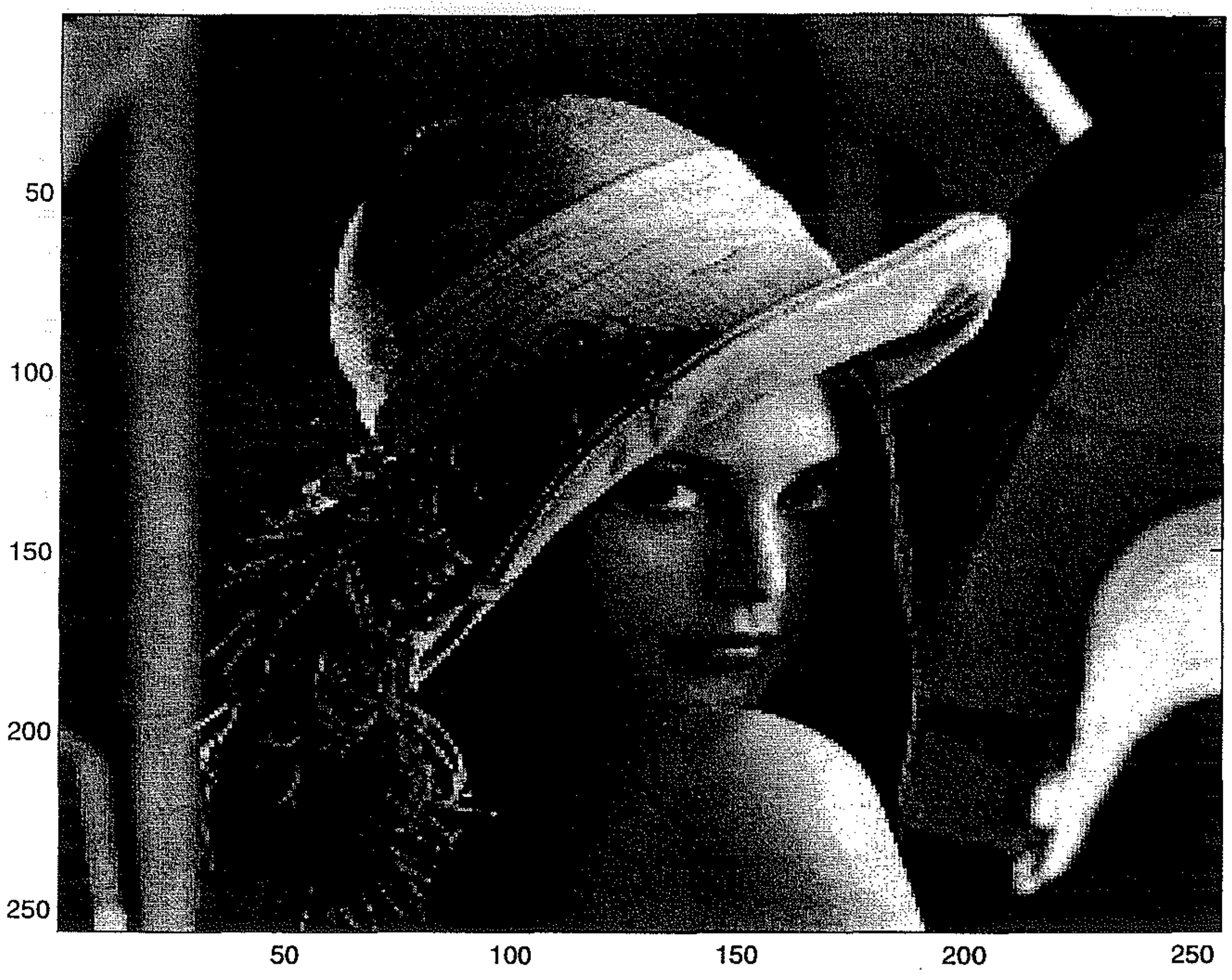
Filter Type 5-3
Level 3 inverse transformed image
MSE = 0.0026



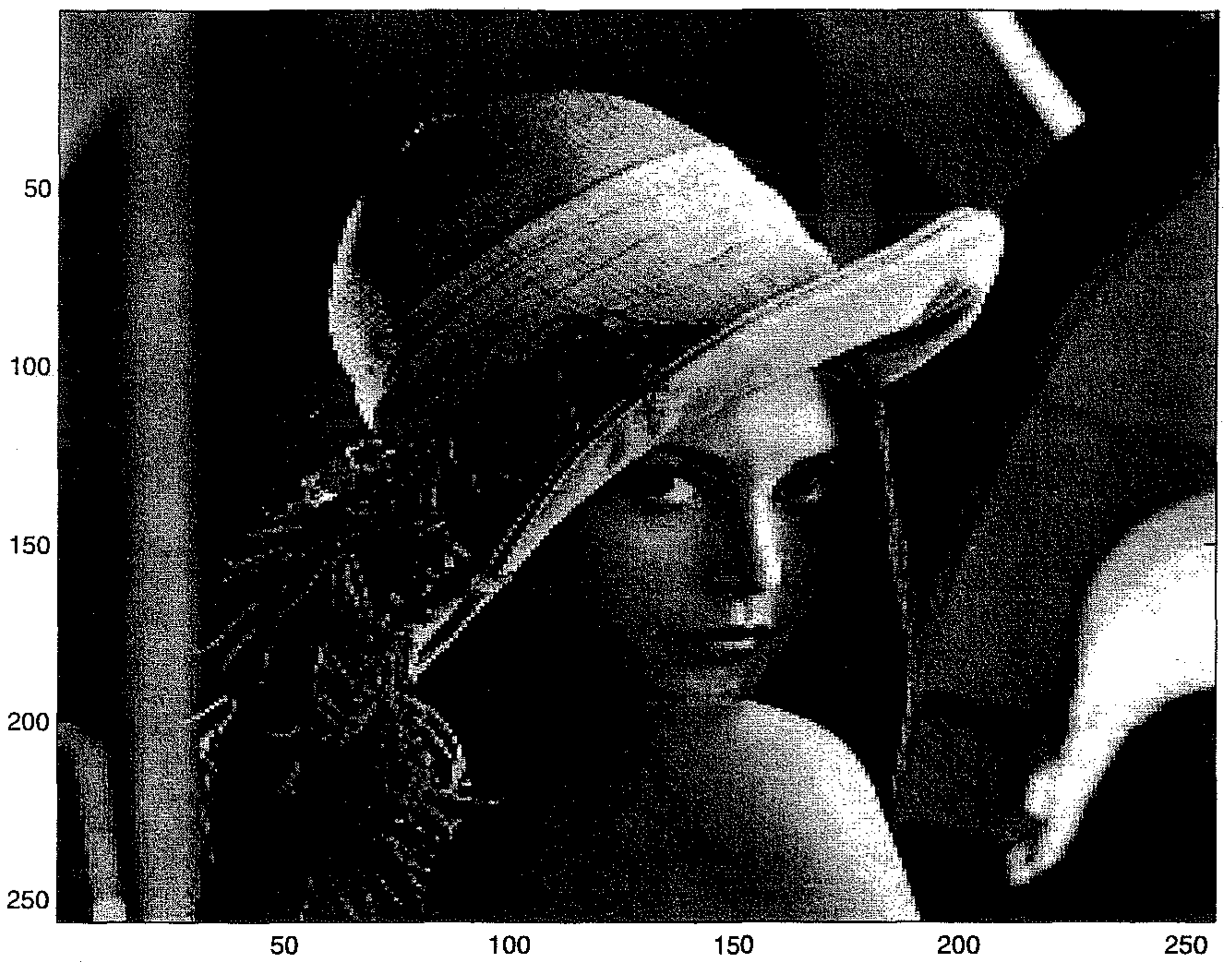
Filter Type 6-10
Level 3 inverse transformed image
MSE = 0.0067



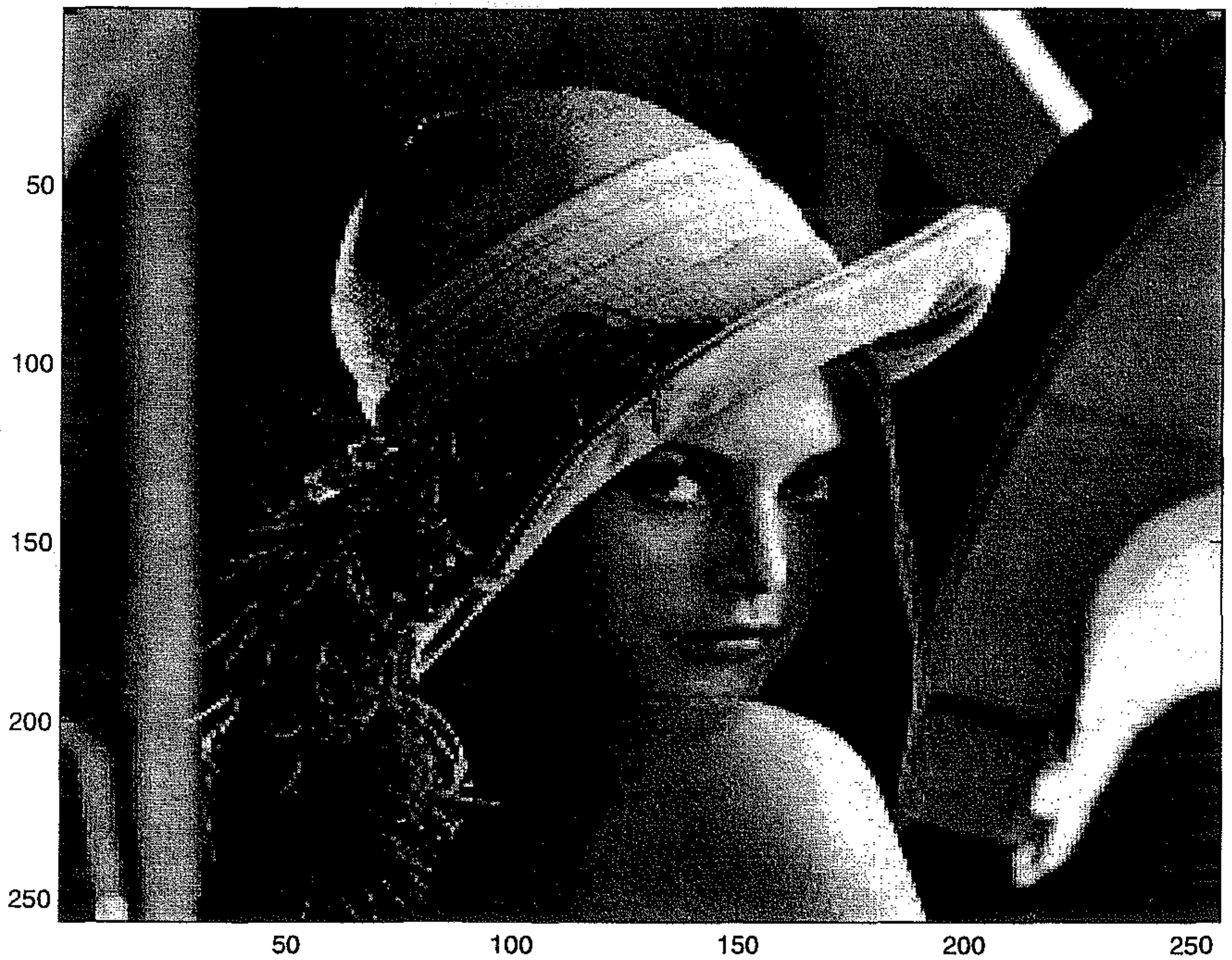
Filter Type 9-3
Level 3 inverse transformed image
MSE = 0.0029



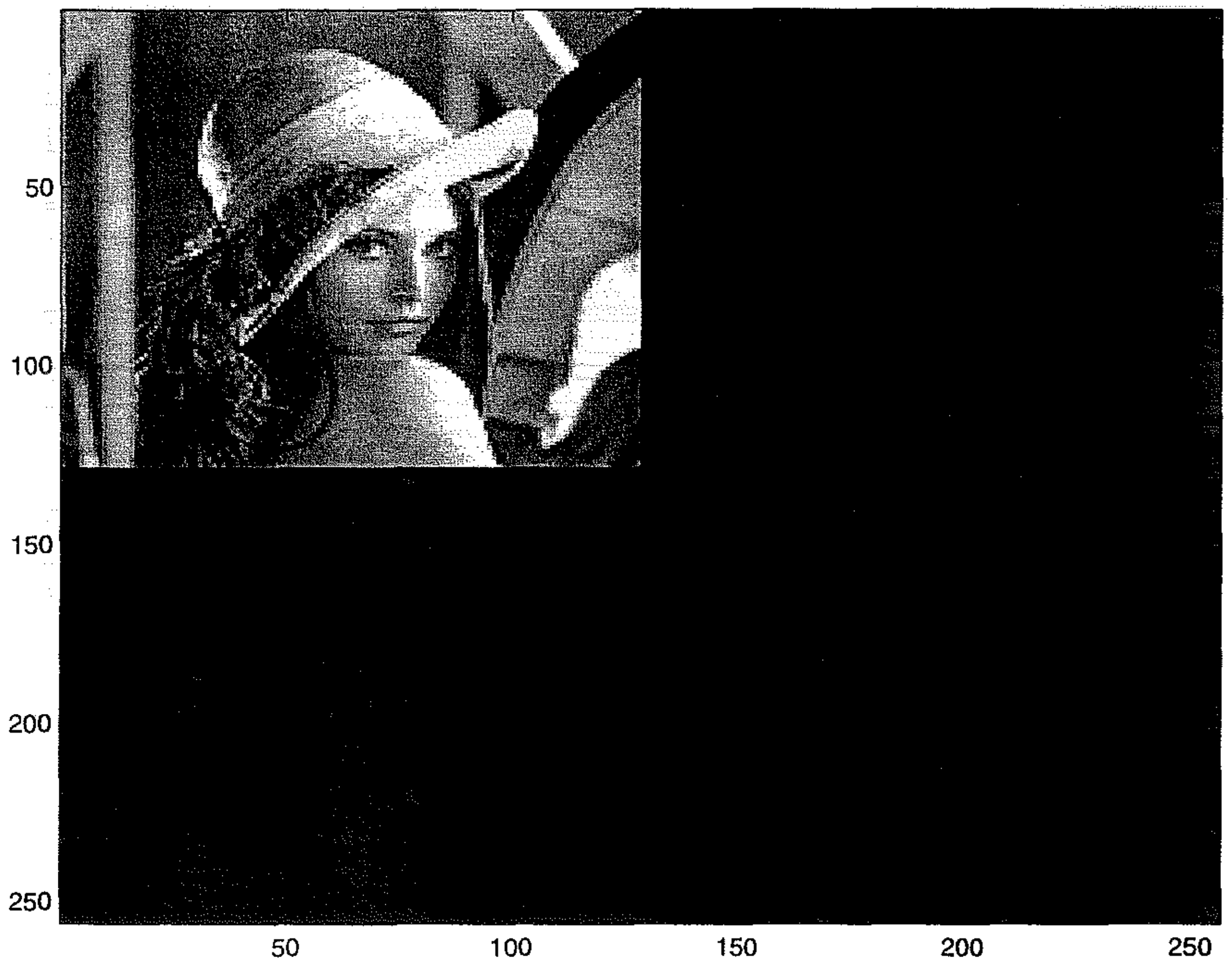
Filter Type 9-7
Level 3 inverse transformed image
MSE = 0.0035



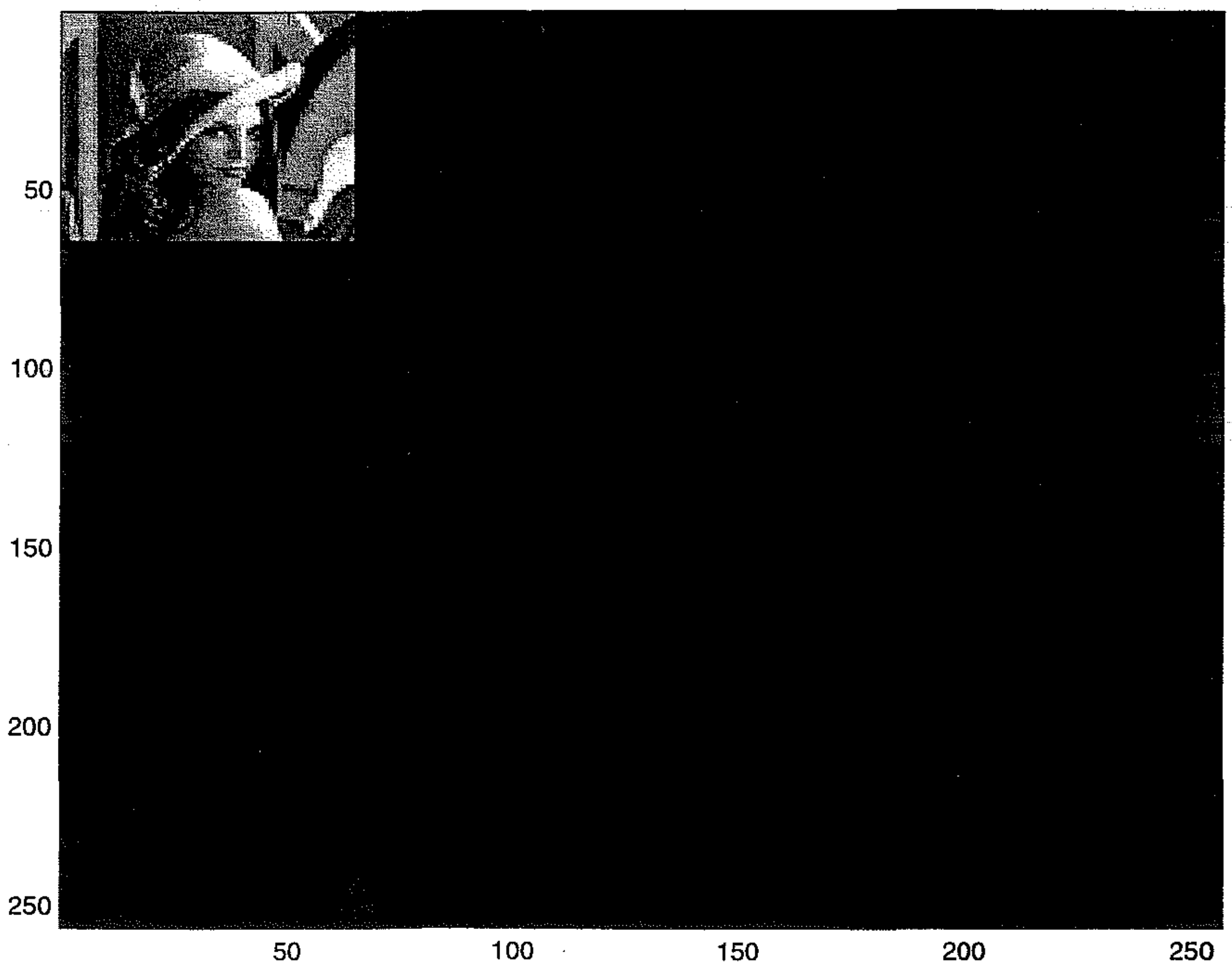
Filter Type 10-18
Level 3 inverse transformed image
MSE = 0.0011



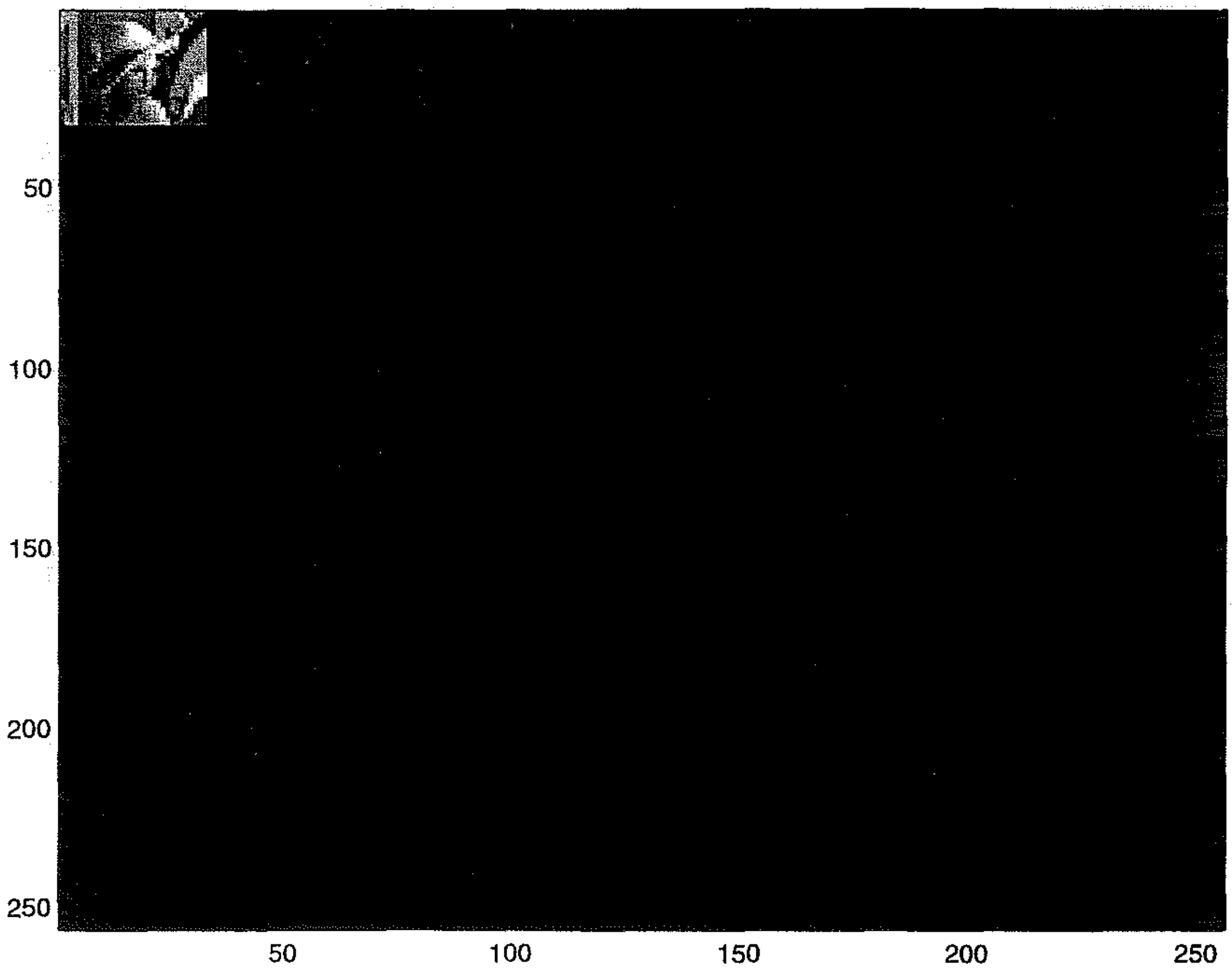
Filter Type 13-11
Level 3 inverse transformed image
MSE = 0.0041



Filter Type 10-18
First octave subband decomposition

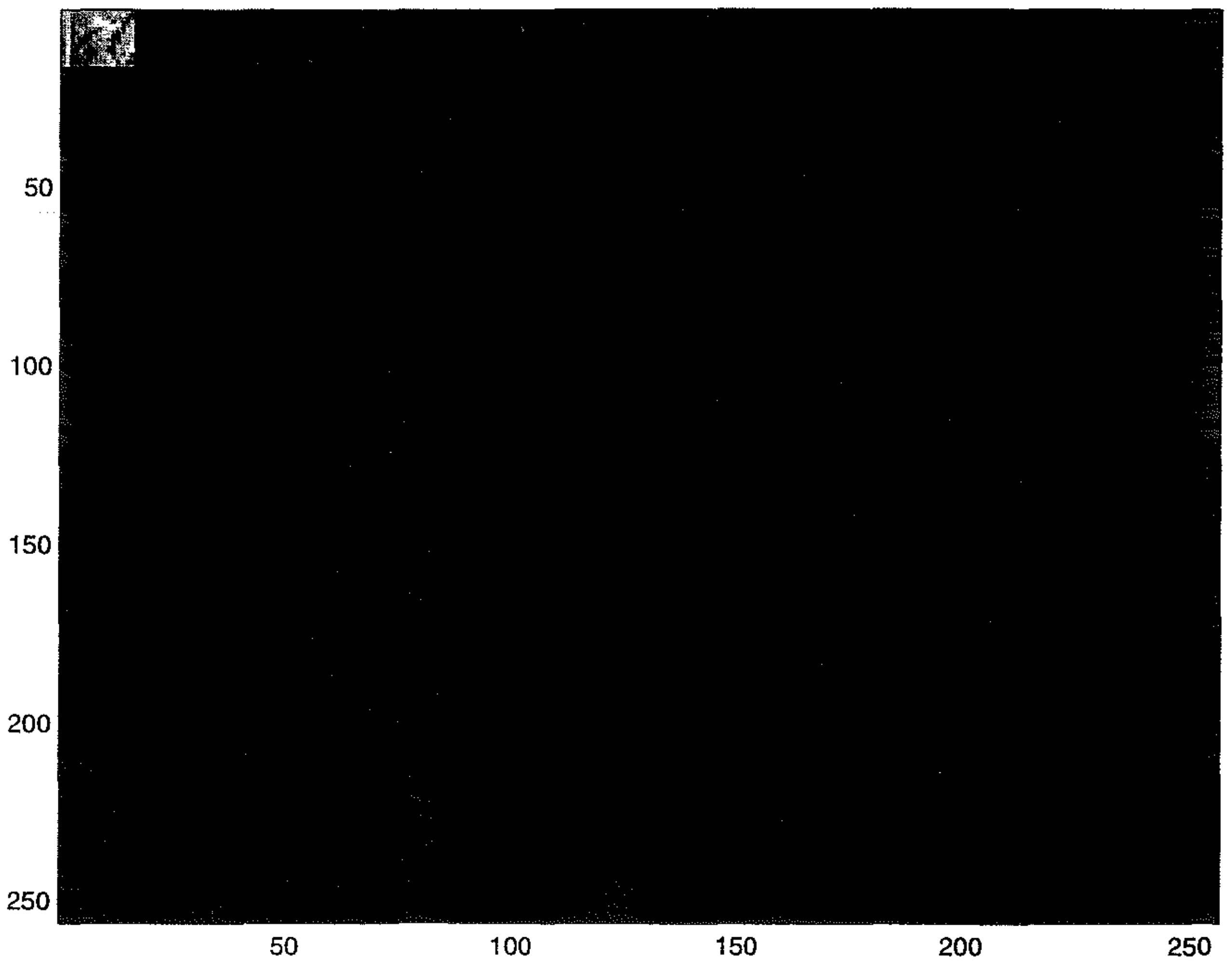


Filter Type 10-18
Second octave subband decomposition

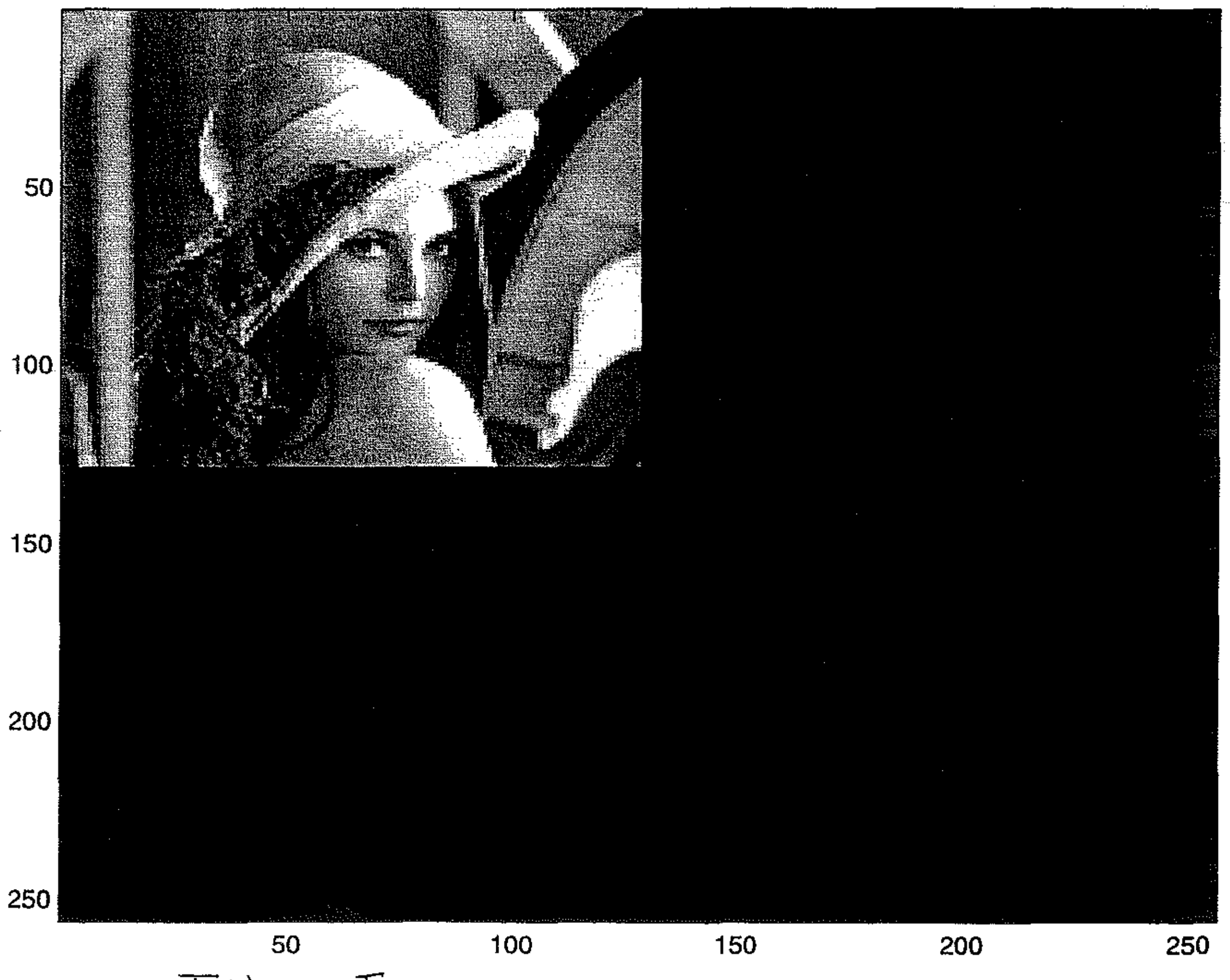


Filter Type 10-18

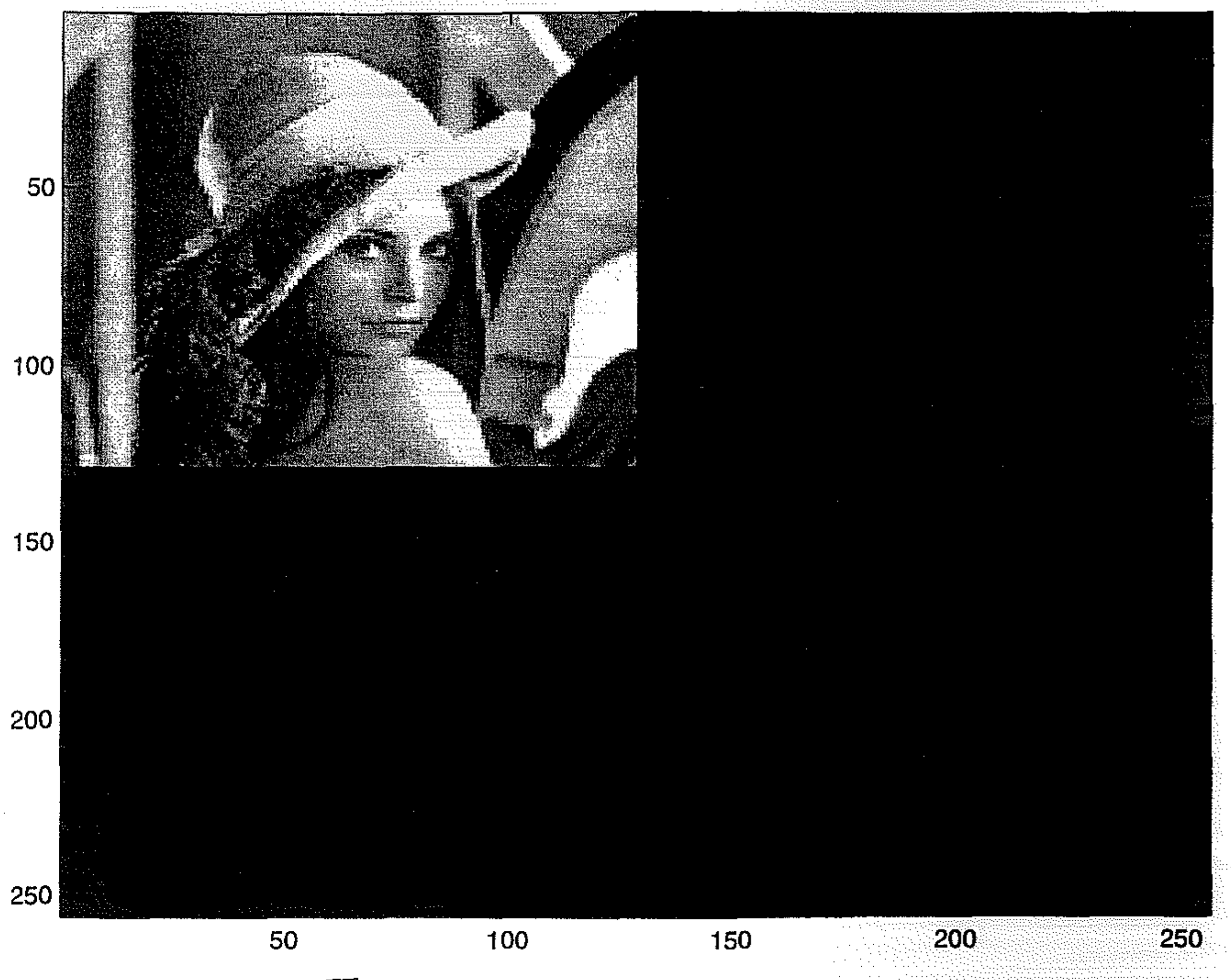
Third octave subband decomposition



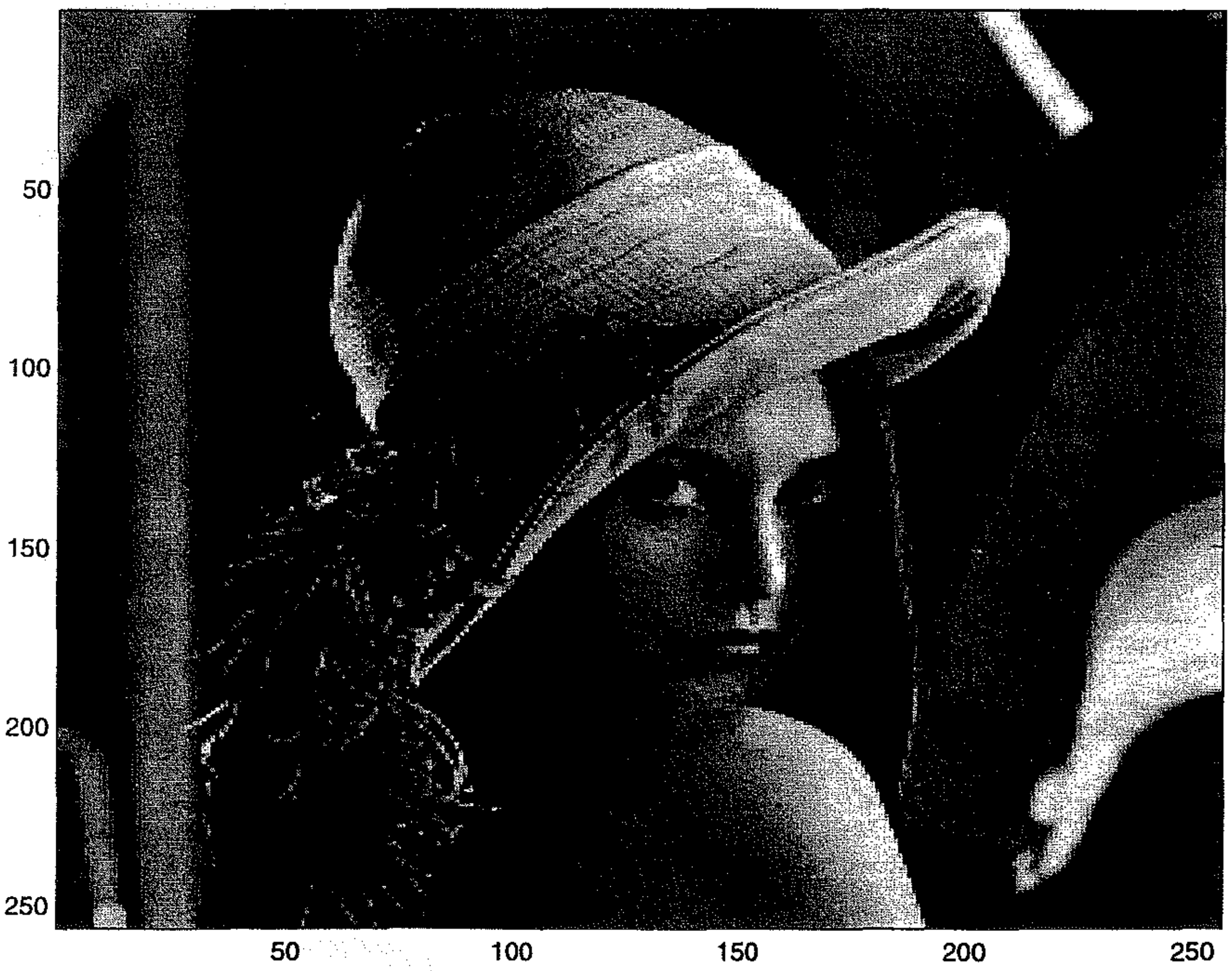
Filter Type 10-18
Forth octave subband decomposition



Filter Type 9-7
First octave subband decomposition



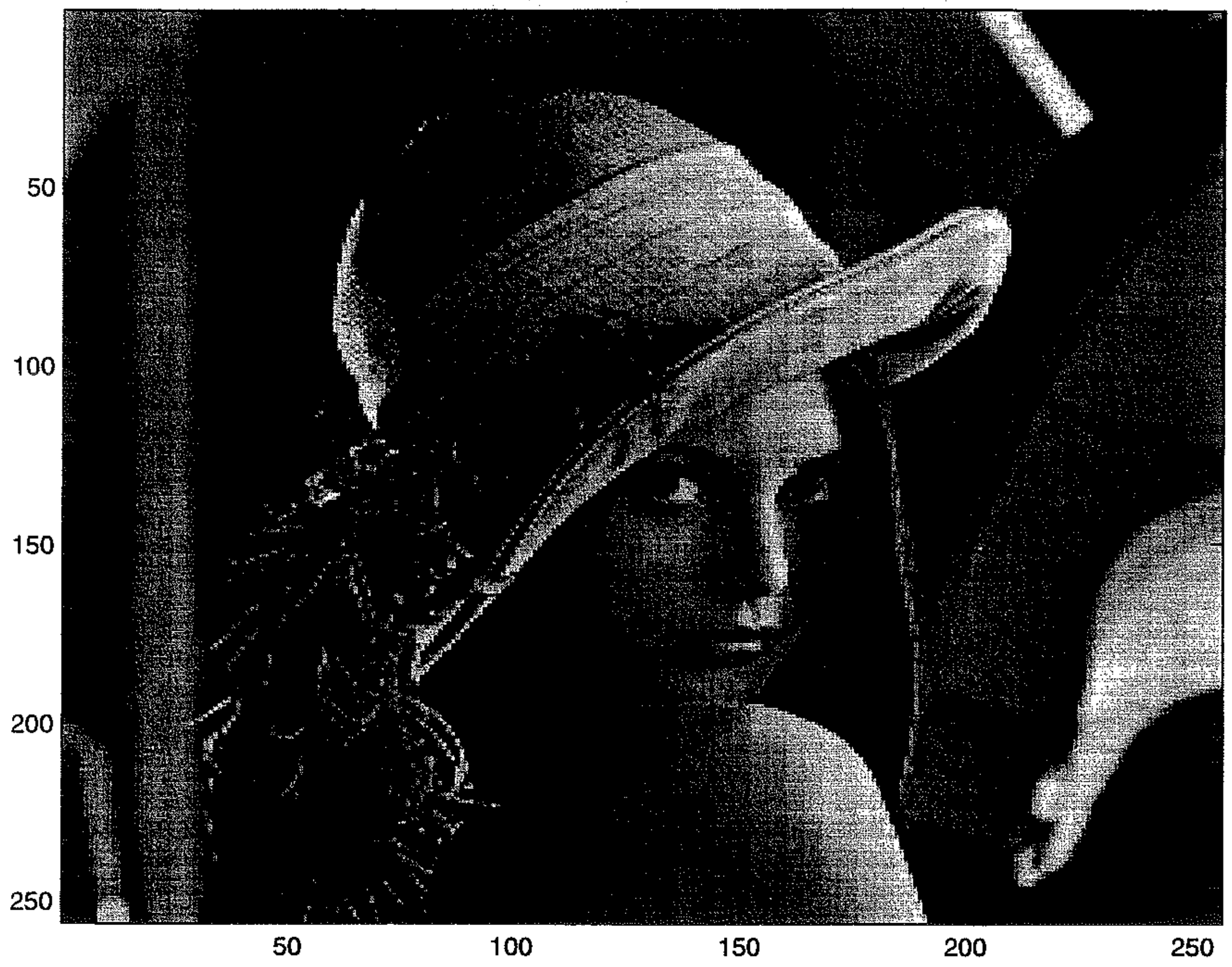
Filter Type 2-2
First octave subband decomposition
Note how high frequency components contain greater amounts of significant coefficients.



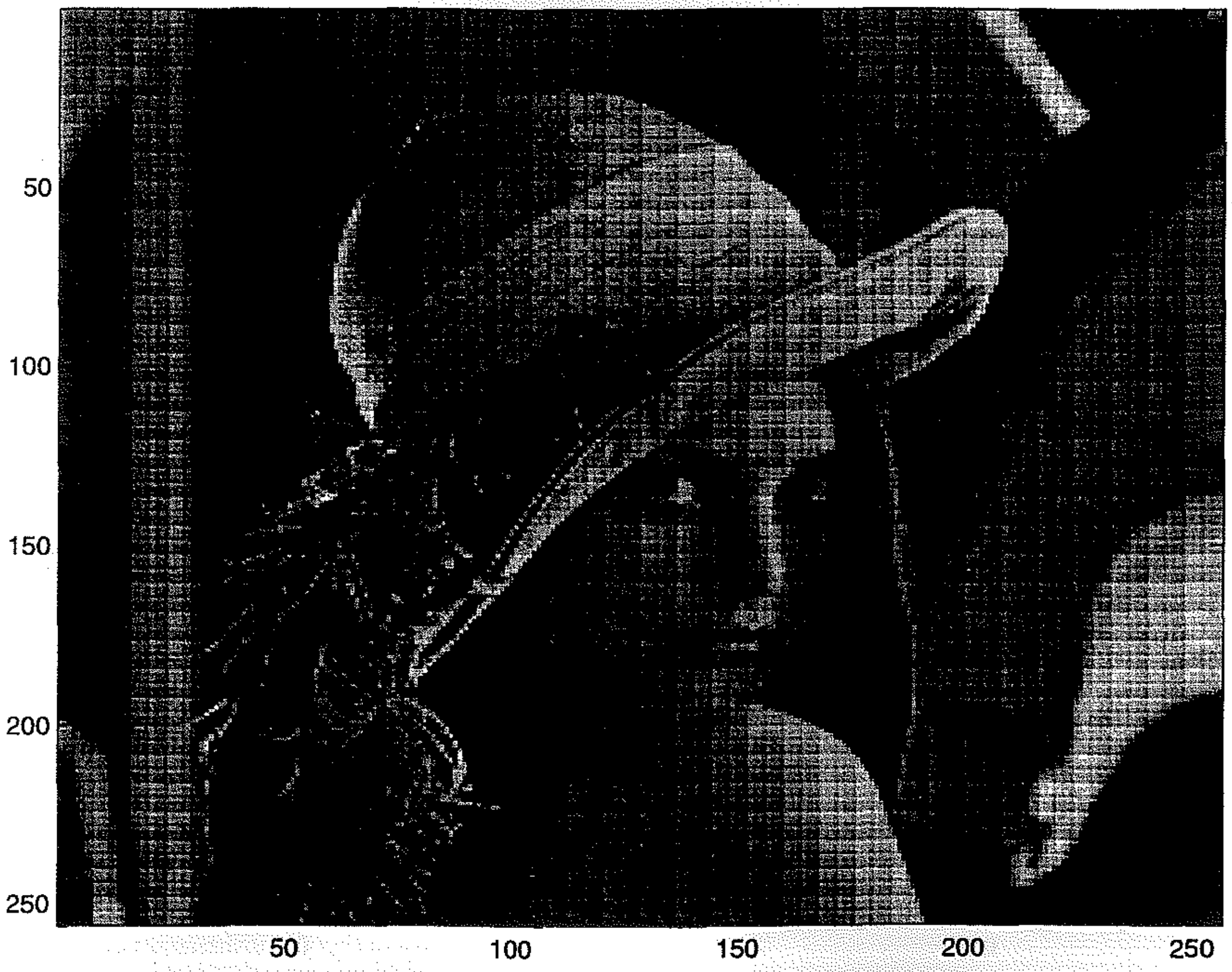
Filter Type 10-18

Fraction part precision for filter and data coefficients is 32 bits.

MSE = 0.0010



Filter Type 10-18
Fraction part precision for filter and data coefficients is 17 bits.
MSE = 0.0596



Filter Type 10-18
Fraction part precision for filter and data coefficients is 8 bits.
MSE = 0.0906