IMAGE CODING USING TWO DIMENSIONAL WAVELET TRANSFORM AND IMPELEMENTING IN FPGA

Submitted
In partial fulfillment of the academic requirements for
the award of

MASTER OF TECHNOLOGY
IN
INTEGRATED CIRCUIT TECHNOLOGY

By
N.VENU GOPALA KRISHNA
06PHMI09

School of Physics
UNIVERSITY OF HYDERABAD
HYDERABAD - 5000046

June 2008

UNIVERSITY OF HYDERABAD

HYDERABAD-5000046

INDIA

DECLARATION

I, N.Venu Gopala Krishna here by declare that the work embodied in this dissertation entitled **"IMAGE CODING USING TWO DIMENSIONAL WAVELET TRANASFORM AND IMPLEMENTING IN FPGA"** submitted to the University Of Hyderabad for partial fulfillment of the degree of **M.Tech in Integrated Circuit Technology** has been carried out by me under the supervision of **Dr.Samrat L. Sabat**, School of Physics, University Of Hyderabad. To the best of my knowledge, this work has not been submitted for any other degree in any University.

Venu Gopala Krishna .N

M.Tech(IC Technology)

Reg.NO: 06PHMI09

UNIVERSITY OF HYDERABAD

HYDERABAD-5000046

INDIA

CERTIFICATE

This is to certify that the project work **"IMAGE CODING USING TWO DIMENSIONAL WAVELET TRANASFORM AND IMPLEMENTING IN FPGA"** carried out by **Mr. N.Venu Gopala Krishna** bearing the Reg.No.06PHMI09 under my guidance in partial fulfillment of the requirements for the award of **Master of Technology in Integrated Circuit Technology** in University of Hyderabad. The matter embodied in this thesis has not been submitted in any other university for the award of any other degree.

Supervisor:

Dr. Samrat L. Sabat,
School of Physics.

Dean
School of Physics.

ACKNOWLEDGEMENTS

This work would not have been complete without the help and support of many people. I am grateful to my supervisor **Dr. Samrat L. Sabat,** University of Hyderabad for his invaluable guidance and support throughout the project work, and more importantly during times of extreme crisis.

I take this opportunity to thank the members of my roommates, and all my friends for their moral support during project work. Their encouragement and support was invaluable in completing this work.

.

ABSTRACT

Image Coding is a project established for storing or sending images using few bits as possible for encoding a complete image. This project produces the compressed image can either be exactly equal to the original image or differ from it in a limited and controlled way.

This project aims at hardware digital system designing using very high speed integrated circuit hard ware description language.

Here, in this project work, for creating wavelet filter bank using wavelet coefficients, two's complement binary multiplication technique is used to multiply the two real numbers. The selected filter bank is bi-orthogonal 3.7, which is suitable for most of the practical applications.

In this work, the input image is converted in to two dimensional matrix and the resulting coefficients at the simulation of hard ware design in VHDL that is the output coefficients in VHDL are converted in to the image using MATLAB tool view the output.

The digital circuit for image coding is generated such that as the thresholding changes according to that compression ratio changes.

CONTENTS

About The CD

The CD-ROM accompanying this thesis includes MATLAB the program code used. It also includes the VHDL code used in this work. For details see the related documents on the CD.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Image

An image is represented as a two dimensional function *f(x, y)* where *x* and *y* are spatial co-ordinates and the amplitude of '*f*' at any pair of coordinates *(x, y)* is called the intensity of the image at that point.

A grayscale image is a function *I (x, y)* of two spatial coordinates of the image plane. The amplitude of *I (x, y)* represents intensity of an image at the point *(x, y)* on image plane. Color image can be represented by three functions, *R (x, y)* for red, *G (x, y)* for green and *B (x, y)* for blue.

An image may be continuous with respect to the *x* and *y* coordinates and also in amplitude. Converting such an image to digital form requires that the coordinates as well as the amplitude to be digitized. Digitizing the co ordinate's values is called sampling. Digitizing the amplitude values is called quantization.

1.2 Types Of Image

There are four types of images:

1. Intensity image
2. Binary images
3. Indexed images
4. R G B images

Most monochrome image processing operations are carried out using binary or intensity images, so our initial focus is on these two image types. Indexed and RGB colour images [1].

1**. Intensity image**: An intensity image is a data matrix whose values have been scaled to represent intentions. When the elements of an intensity image are of class unit8, or class unit 16, they have integer values in the range [0,255] and [0, 65535], respectively. If the image is of class double, the values are floating point numbers. Values of scaled, double intensity images are in the range [0, 1] by convention.

2. **Binary image**: Binary images have a very specific meaning .A binary image is a logical array 0s and 1s. Thus, an array of 0s and 1s whose values are of data class, say unit8, is not considered as a binary image .A numeric array is converted to binary using function logical. If A contains elements other than 0s and 1s.Use of the logical function converts all nonzero quantities to logical 1s and all entries with value 0 to logical 0s.Using relational and logical operators also creates logical arrays. If c is a logical array, this function returns a 1.Otherwise returns a 0. Logical array can be converted to numeric arrays using the data class conversion functions.

3**. Indexed image**: An indexed image has two components: A data matrix integer, *x* and A color map matrix, map. Matrix map is an m*3 arrays of class double containing floating_ point values in the range [0, 1]. The length m of the map is equal to the number of colors it defines. Each row of map specifies the red, green and blue components of a single color. An indexed image uses “direct mapping” of pixel intensity values color map values. The color of each pixel is determined by using the corresponding value the integer matrix x as a pointer in to map. If x is of class double, then all of its components with values less than or equal to 1 point to the first row in map, all components with value 2 point to the second row and so on. If x is of class units or unit 16, then all components value 0 point to the first row in map, all components with value 1 point to the second and so on.

4. **RGB image**: An RGB colour image is an M*N*3 array of color pixels where each colour pixel is triplet corresponding to the red, green and blue components of an RGB image, at a specific spatial location. An RGB image may be viewed as “stack” of three gray scale images that when fed in to the red, green and blue inputs of a color monitor. Produce a colour image on the screen. Convention the three images forming an RGB

color image are referred to as the red, green and blue components images. The data class of the components images determines their range of values. If an RGB image is of class double the range of values is [0, 1].

Similarly the range of values is [0,255] or [0, 65535]. For RGB images of class unit8 or unit 16 respectively. The number of bits use to represents the pixel values of the component images determines the bit depth of an RGB image. For example, if each component image is an 8bit image, the corresponding RGB image is said to be 24 bits deep.

Generally, the number of bits in all component images is the same. In this case the number of possible color in an RGB image is (2^b) ^3, where b is a number of bits in each component image. For the 8bit case the number is 16,777,216 colors**.**

1.3 Block Diagram

General Image Compression system

As shown in figure below image compression systems are composed of two distinct structural blocks: an encoder & a decoder. Image *f(x, y)* is fed into the encoder, which creates a set of symbols from the input data &uses them to represent the image.

Figure1.3 General Image compression system block diagram.

If it lets *X1* & *X2* denote the number of information carrying units in the original & encoded images respectively, the compression that is achieved can be quantified numerically via the compression ratio(*CR*).

$$CR=X1/X2 \quad \text{..... (1.3)}$$

A compression ratio like 4 or 4:1 indicates that the original image has 4 information carrying units for every 1 unit in the compressed data set.

1.4 Why Image Compression?

Statement of the problem: Uncompressed data requires considerable storage capacity and transmission bandwidth. Despite rapid progress in mass-storage density, processor speeds, and digital communication system performance, demand for data storage capacity and data-transmission bandwidth continues to outstrip the capabilities of available technologies [2]. The recent growth of data intensive multimedia-based web applications have not only sustained the need for more efficient ways to encode signals and images but have made compression of such signals central to storage and communication technology.

The performance of these coders generally degrades at low bit-rates mainly because of the underlying block-based Discrete Cosine Transform scheme. More recently, the wavelet transform has emerged as a cutting edge technology, within the field of image compression. Wavelet-based coding provides substantial improvements in picture quality at higher compression ratios.

1.5 Techniques For Compression

Image compression can be lossy or lossless. Lossless compression is sometimes preferred for artificial images such as technical drawings, icons or comics. This is because lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossless compression methods may also be preferred for high value content, such as medical imagery or image scans made for archival purposes. Lossy methods are especially suitable for natural images such as photos in applications where minor, sometimes imperceptible loss of fidelity is acceptable to achieve a substantial reduction in bit rate.

Methods for lossless image compression are:

- Run-length encoding – used as default method in PCX and as one of possible in BMP, TGA, TIFF.
- Entropy coding
- Adaptive dictionary algorithms such as LZW – used in GIFand TIFF
- Deflation– used in PNG, MNG and TIFF

Methods for lossy compression:

- Reducing the color space to the most common colors in the image. The selected colors are specified in the color palette in the header of the compressed image. Each pixel just references the index of a color in the color palette. This method can be combined with dithering to avoid posterization.
- Chroma sub sampling. This takes advantage of the fact that the eye perceives brightness more sharply than color, by dropping half or more of the chrominance information in the image.
- Teansform coding: This is the most commonly used method. A Fourier-related transform such as DCT or the wavelet transform are applied, followed by quantization and entropy coding.
- Fractal compression.

Some of the coding techniques are disscusssed in third chapter in detail.

1.6 What Is Wavelet?

Wavelets are mathematical functions that break up data into different frequency components, and then analyze each component with a resolution matched to its scale [3]. It has advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. Wavelets were developed independently in the fields of mathematics, quantum physics, electrical engineering, and seismic geology. Interchanges between these fields during the last ten years have led to many new wavelet applications such as image compression, turbulence, human vision, radar, and earthquake prediction. This paper introduces wavelets to the interested technical person outside of the digital signal-processing field. it describe the history of

wavelets beginning with Fourier, compare wavelet transforms with Fourier transforms, state properties and other special aspects of wavelets, and finish with some interesting applications such as image compression, musical tones, and de-noising noisy data.

1.7 Motivation and Objectives of this Study

The fundamental idea behind wavelets is to analyze the signal according to scale. Wavelets are functions that satisfy certain mathematical requirements [4] .However, in wavelet analysis, the *scale* parameter plays a special role. Wavelet algorithms process data at different *scales* or *resolutions.* Wavelet analysis is being used for time frequency analysis of signal. The different scale of the transforms gives information about different frequency as scale is inversely proportional to frequency. This makes wavelets interesting and useful for analyzing non stationary signals.

Image compression is an active area of research in signal processing community. Number of techniques is available for compressing the image [4]. But still there is a demand to improve the algorithm such that the compression ratio will be more without distorting the quality of image. Recently wavelet has a promising application in image compression field [3]. Although in algorithm, wavelet can be applied for compressing the image, but due to its heavy computational cost, it is not popular for use in real time or online application. So there is a scope for implementing the wavelet transform in hardware, which will accelerate the compression times to many folds. In recent pasts Field programmable gate array (FPGA) has promising applications in digital signal processing applications. So study of image compression and its possibility for FPGA implementation has been studied in this work.

1.8. Technical Approach and issues

In general, in the designing of the image coding system, we deal with the digital data obtained by reading the image in to the matlab as the input to the system. On this, wavelet analysis is performed it means decomposition and reconstruction along with thresholding before reconstruction is performed. The technical steps involved in the work are i) Studying image compression technique using wavelet transform. This has been

carried out in MATLAB environment, and ii) RTL coding and implementation of the compression algorithm in FPGA. Simulation has been carried out for Virtex 4 FPAGA. The details of these steps are presented in chapter 3 and chapter 4 respectively.

1.8.1 What Is FPGA?

Short for **F**ield-**P**rogrammable **G**ate **A**rray is a programmable logic chip. An FPGA is similar to a Programmable Logic Device (PLD), but whereas PLDs are generally limited to hundreds of gates, FPGAs support thousands of gates. They are especially popular for prototyping integrated circuit designs. Once the design is set, hardwired chips are produced for faster performance.

The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Combining these basic blocks to create the desired circuit creates complex designs.

Individually defining the many switch connections and cell logic functions would be a daunting task. Fortunately, this task is handled by special software. The software translates a user's schematic diagrams or textual hardware description language code then places and routes the translated design. Most of the software packages have hooks to allow the user to influence implementation, placement and routing to obtain better performance and utilization of the device. Libraries of more complex function macros (eg. adders) further simplify the design process by providing common circuits that are already optimized for speed or area.

1.8.2 Why FPGA Is Chosen?

Large-scale integrated circuits (LSI**)** formed the backbone of most of the logic circuits in major systems. Microprocessors, bus/IO controllers, system timers etc were implemented using integrated circuit fabrication technology. Random “glue logic” or interconnects were still required to help connect the large integrated circuits in order to: Generate global control signals for resets etc., Data signals from one subsystem to another sub system.

Systems typically consisted of few large scale integrated components and large number of SSI (small scale integrated circuit) and MSI (medium scale integrated circuit) components.

Initial attempt to solve this problem led to development of Custom ICs, which were to replace the large amount of, interconnect. This reduced system complexity and manufacturing cost, and improved performance. However, custom ICs has their own disadvantages. They are relatively very expensive to develop, and delay introduced for product to market (time to market) because of increased design time. There are two kinds of costs involved in development of Custom Ics.A tradeoff usually exists between the twocosts: 1.Cost of development and design 2. Cost of manufacture

FPGAs were introduced as an alternative to custom ICs for implementing entire system on one chip and to provide flexibility of reporogramability to the user. Introduction of FPGAs resulted in improvement of density relative to discrete SSI/MSI components (within around 10 xs of custom ICs). Another advantage of FPGAs over CustomICs is that with the help of computer aided design (CAD) tools circuits could be implemented in a short amount of time (no physical layout process, no mask making, no IC manufacturing).

1.8.3 Hardware Implementation

The design flow comprises the following steps: design entry, design synthesis, design implementation, and device programming. Design verification includes both

functional verification and timing verification takes places at different points during the design flow. Design Entry includes the fallowing steps in it.

1. Creating a project Navigator project.
2. Creating files and add them to the project created.
3. Adding existing files to the project.
4. Assigning constraints such as timing constraints, pin constraints, and area constraints.

Functional Verification has below steps:

1. Before synthesis, running behavioral simulation.
2. After device programming, running in circuit verification.

Design synthesis includes the following steps:

1. Implement the design includes the steps a. Translate b. Map c. Place and route.
2. Synthesize and implement the design until design requirements are met.

Timing Verification is running static timing analysis at after Map and place and route.

Device programming is creating programming file and down loading to the selected device.

1.9 Thesis Scope

It should be emphasized that this thesis is restricted to study and implementation of wavelet technique for compressing an image

1.10 Thesis Structure

This section briefly outlines the contents of the chapters. The dissertation is organized into five chapters as in the following manner. A list of all reference sources consulted is given chapter wise.

CHAPTER 1 is an introductory chapter. This describes the types of images, general image compression system, need of FPGA, and the motivation behind solving such a problem.

CHAPTER 2 presents a brief description of fft, stft, overview of dwt, convolution and types of convolution. It also briefly discusses the filter bank, all available filter banks.

CHAPTER 3 handles the important topic of this work. Different coding techniques are discussed in this chapter. Lossless coding techniques, lossy coding techniques are discussed briefly and image coding using dwt concept is explained in this in detail.

CHAPTER 4 tells about hardware realization it means FPGA design flow, architecture used for VHDL and Matlab, results obtained, comparisons of results, applications.

CHAPTER 5 Conclusions and recommendation for further work/efforts to implement the hardware design are drawn in this chapter.

References

[1].Digital Image Processing using MATLAB--Rafael C. Gonzalez, Richard E. Woods Steven C. Eddies.

[2]. Ahmed, N., Natarajan, T., and Rao, K. R. Discrete Cosine Transform, *IEEE* Trans. Computers, vol. C-23, Jan. 1974, pp. 90-93.

[3].Paul S. Addison, The Illustrated Wavelet Transform Handbook, Institute of Physics, 2002, ISBN 0-7503-0692-0.

[4].Dubieties, I. (1992), Ten lectures on wavelets, CBMS-NSF conference series in applied mathematics. SIAM Ed.

CHAPTER 2

WAVELET TRANSFORM

2.1 Fast Fourier Transform

A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT). FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers. Let $x_0 ... x_{N-1}$ be real numbers. The DFT is defined by the formula 2.1.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-(2\prod j/N)nk} \quad \text{..... (2.1)}$$

$$k = 0,1,.....N-1$$

Evaluating these sums directly would take O (N^2) arithmetical operations. An FFT is an algorithm to compute the same result in only O (N log N) operations. In general, such algorithms depend upon the factorization of N, but there are FFTs with O (N log N) complexity for all N, even for prime N.

2.2 Short Time Fourier Transform

The short-time Fourier transform (STFT), or alternatively short-term Fourier transform, is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.

A. Continuous-time STFT

Simply described, in the continuous-time case, the function to be transformed is multiplied by a window function which is nonzero for only a short period of time. The

Fourier transform (a one-dimensional function) of the resulting signal is taken as the window is slid along the time axis, resulting in a two-dimensional representation of the signal. Mathematically, this is written as:

$$\text{STFT } \{x(t)\} = X(T, w) = \int_{-\infty}^{\infty} x(t)w(t-T)e^{-jwt}\,dt \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2.2a)}$$

Where x (t) is a continuous signal and w (t) is short time window, commonly a Hann window or Gaussian hill centered around zero, and $x(t)$ is the signal to be transformed. $X(T, w)$ is essentially the Fourier Transform of $x(t)w(t-\tau)$, a complex function representing the phase and magnitude of the signal over time and frequency. Often phase unwrapping is employed along either or both the time axis, τ and frequency axis, w, to suppress any jump discontinuity of the phase result of the STFT. The time index τ is normally considered to be *slow* time and usually not expressed in as high resolution as time *t*.

B. Discrete-time STFT

In the discrete time case, the data to be transformed could be broken up into frames. Each frame is Fourier transformed, and the complex result is added to a matrix, which records magnitude and phase for each point in time and frequency. This can be expressed as:

$$\text{STFT } \{x(n)\} = X(m, w) = \sum_{-\infty}^{\infty} x(n)w(n-m)e^{-jwn} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2.2b)}$$

Where $x(n)$ is a discrete signal and $w(n)$ is short time window.

2.3 Wavelet Transform

Wavelet Transform represents the next logical step: a windowing technique with Variable-sized regions. Wavelet analysis allows the use of long time intervals where it want more precise low-frequency information, and shorter regions where it want high-frequency information.

Figure 2.3a Wavelet Transform

The figure2.3b shows contrast with the time-based, frequency-based, and STFT views of a signal:

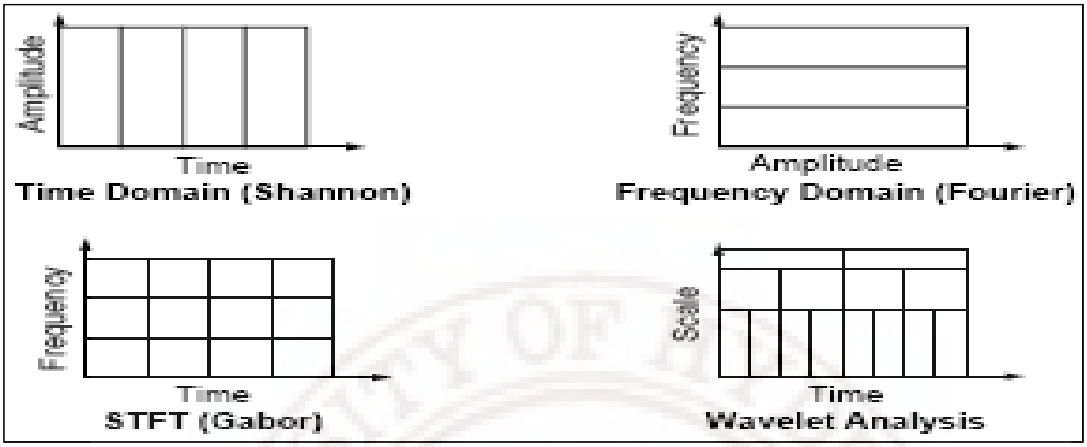


Figure 2.3b Wavelet, Fourier, STFT amplitude graphs

It may have noticed that wavelet analysis does not use a time-frequency region, but rather a time-scale region.

2.4 What Is Wavelet Analysis?

A wavelet is a waveform of effectively limited duration that has an average value of zero. By comparing wavelets with sine waves, which are the bases of Fourier analysis. Sinusoids do not have limited duration, they extend from minus to plus infinity. And where sinusoids are smooth and predictable, wavelets tend to be irregular and asymmetric.

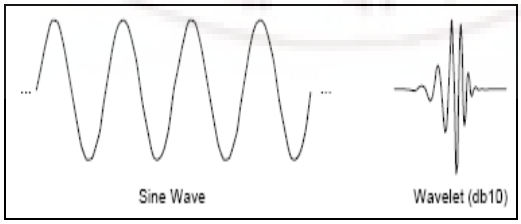


Figure 2.4 Sine wave and Wavelet (db 10)

Fourier analysis consists of breaking up a signal into sine waves of various frequencies. Similarly, wavelet analysis is the breaking up of a signal into shifted and scaled versions

of the original (or mother) wavelet. Just looking at pictures of wavelets and sine waves, it can see intuitively that signals with sharp changes might be better analyzed with an irregular wavelet than with a smooth sinusoid, just as some foods are better handled with a fork than a spoon. It also makes sense that local features can be described better with wavelets that have local extent.

Wavelet analysis is capable of revealing aspects of data that other signal analysis techniques miss aspects like trends, breakdown points, discontinuities in higher derivatives, and self-similarity. Furthermore, because it affords a different view of data than those presented by traditional techniques, wavelet analysis can often compress or de-noise a signal without appreciable degradation. One major advantage afforded by wavelets is the ability to perform local analysis, that is, to analyze a localized area of a larger signal Wavelet analysis can be applied to two-dimensional data (images*)* and, in principle, to higher dimensional data.

2.5 The Continuous Wavelet Transform

Mathematically, the process of Fourier analysis is represented by the Fourier transform:

$$F(w) = \int_{-\infty}^{\infty} f(t) e^{-jwt}\, dt \quad \text{..... (2.5a)}$$

The results of the transform are the Fourier coefficients F(w), which when multiplied by a sinusoid of frequency w yields the constituent sinusoidal components of the original signal. Graphically, the process looks like:

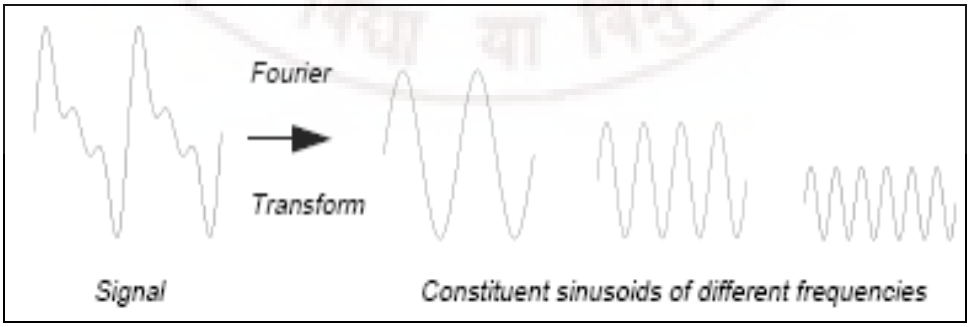


Figure 2.5a Fourier Transform

Similarly, the continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function ψ

$$C(scale, position) = \int_{-\infty}^{\infty} f(t)\Psi(scale, position, t)\, dt$$ ……..……eq (2.5b)

The result of the CWT is a series many wavelet coefficients **C**, which are a function of scale and position**.** Multiplying the signal by the appropriately scaled and shifted wavelet yields the constituent wavelets of the original signal:

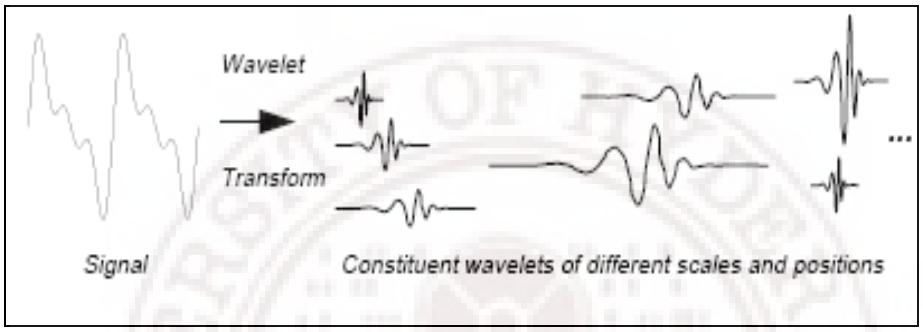


Figure 2.5b Wavelet Transform scaling

2.6 Scaling and Shifting

a. Scaling

It is known that wavelet analysis produces a time-scale view of a signal and now, it is scaling and shifting wavelets.. Scaling a wavelet simply means stretching or compressing it.. The smaller the scale factor, the more "compressed" the wavelet.

b. Shifting

Shifting a wavelet simply means delaying or hastening. Mathematically, delaying a functioṇ ψ (t) by *k* is represented by ψ (t-k), shown in figure 2.8.

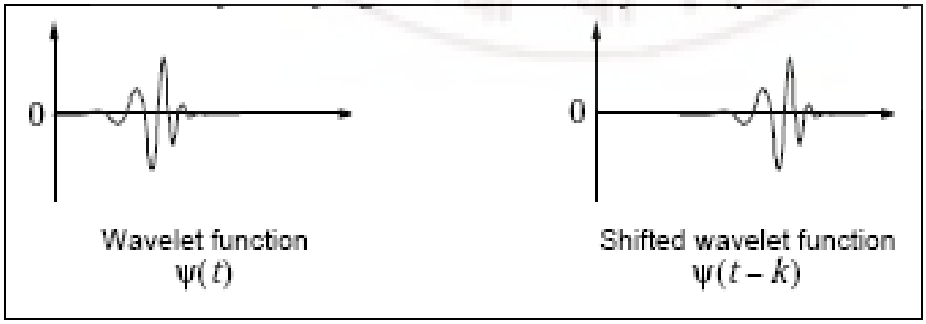


Figure 2.6 Shifting

2.7 The Discrete Wavelet Transform

Calculating wavelet coefficients at every possible scale is a fair amount of work, and it generates a lot of data. If it chooses only a subset of scales and positions at which to make our calculations is: if it chooses scales and positions based on powers of two—so-called dyadic scales and positions—then this analysis will be much more efficient and just as accurate. It obtains such an analysis from the discrete wavelet transform (DWT). The DWT of a signal *x* is calculated by passing it through a series of filters. First the samples are passed through a low pass filter g with impulse response *g* resulting in a convolution .The signal is also decomposed simulta -neously using a high-pass filter *h is* shown in equations 2.9.

$$y(n) = x(n) * g(n) = \sum_{-\infty}^{\infty} x(k)g(n-k) \quad \text{..... (2.7a)}$$

$$ylow(n) = x(n) * g(n) = \sum_{-\infty}^{\infty} x(k)g(2n-k) \quad \text{..... (2.7b)}$$

$$yhigh(n) = x(n) * g(n) = \sum_{-\infty}^{\infty} x(k)h(2n-k) \quad \text{..... (2.7c)}$$

The mother wavelet basis function of DWT is defined as

$$\psi_{j,k}(t) = 2^{\frac{j}{2}} \psi\left(2^{j} t - KT\right) \quad \text{.....(2.7d)}$$

Where j coefficient of time translation is K is coefficient of scale. The basis functions can be obtained for different integer values of j and K.

An efficient way to implement this scheme using filters was developed in 1988 by Mallat. The Mallat algorithm is in fact a classical scheme known in the signal processing community as a two-channel sub band coder. This filtering algorithm yields a fast wavelet transfor, *a* box into which a signal passes, and out of which wavelet coefficients quickly emerge.

2.8 Convolution

Convolution is a simple mathematical operation, which is fundamental to many common image-processing operators. Convolution provides a way of “multiplying together” two arrays of numbers, generally of different sizes, but of the same dimension,

to produce a third array of numbers of the same dimension. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image-processing context, the input arrays are normally just a gray level image. The second array is usually much smaller, and is also two-dimensional, and is known as the kernel. Figure 2.10 shows an example image and kernel that it will use to illustrate convolution.

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Figure2.8.Convolution

Figure 2.8 an example small image (left) and kernel (right) to illustrate convolution. The labels within each grid square are used to identify each square. The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together. So, in our example, the value of the bottom right pixel in the output image will be given by equation 2.8a:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23} \quad \text{.....(2.8.a)}$$

If the image has *M* rows and *N* columns, and the kernel has *m* rows and *n* columns, then the size of the output image will have *M* - *m* + 1 rows, and *N* - *n* + 1 columns.

Mathematically the convolution can be written as:

$$O(i,j) = \sum_{k=1}^{m} n \sum_{l=1} I(i+k-1, j+l-1) K(k,l) \quad \text{.....(2.8.b)}$$

Where *i* varies from *1 to* M - m + *1* and *j* varies from *1 to* N - n + *1*. Many implementations of convolution produce a larger output image than this because they relax the constraint that the kernel can only be moved to positions where it fits entirely within the image. Instead, these implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel overlaps the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the image, it is necessary to *find* input pixel values for places where the kernel extends off the end of the image. Typically pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. Therefore in general if it is using a convolution implementation that does this, it is better to clip the image to remove these spurious regions. Removing *n* - 1 pixel from the right hand side and *m* - 1 pixel from the bottom will fix things. Convolution can be used to implement many different operators, particularly spatial filters and feature detectors. Examples include Gaussian smoothing and the Sobel edge detector.

2.8.1 Continuous Convolution

The behavior of a linear, continuous-time, time-invariant system with input signal *x (t)* and output signal *y (t)* is described by the *convolution* integral

$$y(t) = \int_{-\infty}^{\infty} h(v) x(t-v) dv \quad \text{.....(2.8.1)}$$

The signal *h (t)* is the response of the system to a unit impulse input. To compute the output *y (t)* at a specified *t*, first the integrand *h (v) x (t - v)* is computed as a function of *v*. Then integration with respect to *v* is performed, resulting in y *(t).*

These mathematical operations have simple graphical interpretations. First, plot *h (v)* and the "flipped and shifted" *x (t - v)* on the *v* axis, where *t* is fixed. Second, multiply the two signals and compute the signed area of the resulting function of *v* to obtain *y (t).* These operations can be repeated for every value of *t* of interest. To explore graphical convolution, select signals *x (t)* and *h (t)* from the provided examples below, or use the mouse to draw it's own signal or to modify a selected signal. Then click at a desired value of *t* on the first *v* axis. After a moment, *h (v)* and *x(t - v)* will appear. Drag the *t* symbol along the *v* axis to change the value of *t*. For each *t*, the corresponding integrand *h (v) x (t - v)* and the output value *y (t)* will be displayed in their respective windows.

2.8.2 Discrete Convolution

The behavior of a linear, time-invariant discrete-time system with input signal *x[n]* and output signal *y[n]* is described by the convolution sum.

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]\, x[n-k] \quad \text{..... (2.8.2)}$$

The signal *h [n]*, assumed known, is the response of the system to a unit-pulse input. The convolution summation has a simple graphical interpretation. First, plot *h[k]* and the "flipped and shifted" *x [n - k]* on the *k* axis, where *n* is fixed. Second, multiply the two signals to obtain a plot of the summand sequence indexed by *k*.Summing the values of this sequence with respect to *y*ields *y[n]*. These operations can be repeated for every value of *n* of interest. The convolution m-point signal and n-point signal results in m+n-1 points.

To explore graphical convolution, select signals *x [n]* and *h [n]* from the provided examples below, or use the mouse to draw it's own signal or to modify a selected signal. Then click at a desired value of *n* on the first *k* axis. After a moment, *h [k]* and *x [n - k]* will appear. Drag the *n* symbol along the *k* axis to change the value of *n*. For each *n*, the

corresponding summand *h [k]x [n - k]* and output value y*[n]* will be displayed in their respective windows.

2.8.3 Two –D Convolution

The convolution for the two dimensional data which contains horizontal and vertical values is called two-dimensional convolution. In this, there are two types of convolutions, one is row-wise and second one is column-wise.

A. Row-wise Convolution:

Vertical extension is one in which columns are extended symmetrically by particular number. This number is generally length of the filter less one. The example for this is *[a1 a2 a3]* row by 5 is *"a2 a3 a3 a2 a1"a1 a2 a3"a3 a2 a1 a1 a2*. Then these values are sent to the filter and convolved values are selected from the lower Limit (length of the filter) to the upper limit (length of the extended row).

B. Column-wise Convolution:

Horizontal Extension is one in which the rows are extended symmetrically by a particular number that is length of the filter length less one. Example for this is *[a1 a2 a3; a4 a5 a6] by 3 is [a4 a5 a6; a4 a5 a6; a1 a2 a3; "a1 a2 a3 a; a4 a5 a6;"a4 a5 a6; a1 a2 a3; a1 a2 a3]*.Then these values are sent to the filter and convolved values are selected from the lower limit to upper limit like row-wise convolution.

2. 9 Filter Bank

A filter bank is a structure that decomposes a signal in to a collection of sub signals. Depending on the application, these sub signals help emphasize specific aspects of the original signal or may be easier to work with than the original signal. *X* is Output. In the system shown in above figure 2.9, the incoming signal *x (n)* is spilt in to low and high frequency channels by a pair of filters *H,G* respectively. Both filter outputs are then decimated by a factor of 2, resulting in the pair *X (n)* of so called Sub bands. It can be written the following equations describing the in the z-domain the behavior of a complete two channel analysis synthesis system [2].

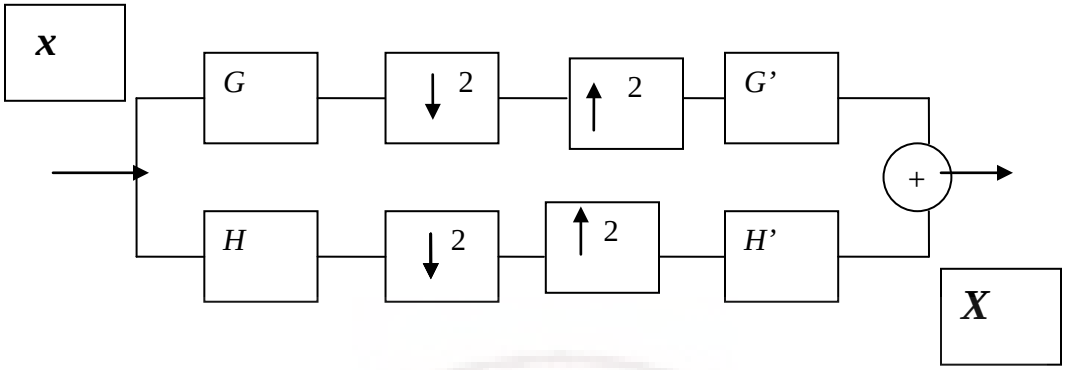


Figure 2.9 Filter Bank Explanation

.2X(n) = [h(z)h'(z)+g(z)g'(z)]+[h(-z)h'(z)+g(-z)g'(z)]x(-z) *(1)*

In this expression, the second term represents the effects of aliasing. this suggests that it can remove aliasing effects by choosing *H, G, H'*, and *G'* in such a way that this second term is zero: *h (-z) h' (z) +g (-z) g' (z) =0* *(2)*

A linear time invariant system with transfer function

T (z) = X (z)/x (z)=1/2(h (z) h' (z) +g (z) g' (z)) *(3)*

Because of the perfect reconstruction Requirement, T (z) should represent a pure delay:$T(z) = z^{-k}$, for any non-negative integer k. Solving the pair of equations (2) and (3) for h' and g' gives $N= z^{-k}$ *h'(z)=2Ng(-z)/D(z), g'(z)=-2Nh(-z)/D(z)With D (z)=h (z) g (-z)-h (-z) g (z).*Now it required that *H'* and *G'* are FIR filters *H* and *G*.This implies that *D* (z) has the form of time shift:*D* (z)=*C* (z^{-L}), for some real *C* and integer *L*.Without loss of generality, it may put *C*=2, and *K=L*, resulting in *h' (z) =g(-z),*

g'(z)=-h(-z) *(4)*

And *h (z) g(-z)- h(-z)g(z)=2(z^{-L})* *(5)*

So, the relationship of filter G and the basic filter *H* is sown below. A clear idea is to make the high pass filter G a frequency –shifted version of the low pass filter *H*, i.e. *G (w)=H (w+pi),* or in *z* transform language,

$g(z)=h(-z)$ (6)

Combining (6) and (5), it get $h2(z)- h^2(-z)= 2(z^{-L})$,Perfect reconstruction is possible if replace(6) by $h(z)=-h(-z^{-L})$

z^{-N} for suitably chosen odd integer $N>0$. (7)

This relation between G and H, in the form of matrix

$h(z)=[h(z)\ h(-z);g(z)\ g(-z)]$ (8)

Orthogonal, in the sense that $h^T(z^{-1})\ h(z) =2I$ for all z.The delay factor z-N has to be present in order to make a casual filter, given that H is causal. To this end, $N+1$ should equal the order of filter H. Filters based on the relations (4),(5) and are called conjugate quadrature filters. Combining (7) with (5), get, given that is odd ,and putting $L=N$,$h(-z)h(-z^{-1})+h(z)h(z^{-1}) = 2$, $or|\ H(w)|^2+|\ H(w+pi)|^2=2$. Where pi = 180 degrees As an example of a Quadrature Filter structure, consider the case of filter order 4, i.e. For N=3, given $h(z)$ = c0+c1z-1+c2z-2+c3z-3t is easily verified using (4),(5) and (7) that remaining 3 filters should be $g(z) = c3-c2z^{-1}+ c1z^{-2}- c0z^{-3,}h'(z) = c3+ c2z^{-1}+ c1z^{-2}+ c0z^{-3,}g'(z) = -c0+ c1z^{-1}- c2z^{-2}+ c3z^{-3}$Also it is found that using (5), that

$C0^2+ C1^2+ C2^2+ C3^2=2$ and $coc2+c1c3 = 0$ (9)

2.10 Length-4 Scaling Coefficient Vector

For the length 4-coefficient vector sequence, there is one degree of freedom or one parameters that gives all the coefficients that satisfy the required conditions [1].

$h(0)+h(1)+h(2)+h(3) = 2^{1/2}$, $h^2(0)+ h^2(1)+ h^2(2)+ h^2(3) = 1$ and $h(0)h(2)+h(1)h(3) = 0$ letting the parameters be the angle D ,the become

$h(0) = (1-\cos(D)+\sin(D))/k$

$h(1) = (1+\cos(D)+\sin(D))/k$

$h(2) = (1+\cos(D)-\sin(D))/k$

$h(3) = (1-\cos(D)-\sin(D))/k$. Where $k= 2x2^{1/2}$.

These equations also give the length-2 haar coefficients for D=0, 90,270 degrees and a degenerate condition for $D=180$.

2.11 Length-6 Scaling Coefficient Vector

For a length 6 coefficient sequence *h(n),*the two parameters are defined as *D* and *E* and resulting coefficients are[1] :

$h(0) = [(1+\cos(D)+\sin(D))(1-\cos(E)-\sin(E))+2\sin(E)\cos(D)]/k1$

$h(1) = [(1-\cos(D)+\sin(D))(1+\cos(E)-\sin(E))-2\sin(E)\cos(D)]/k1$

$h(2) = [1+\cos(D-E)+\sin(D-E)]/k2$

$h(3) = [1+\cos(D-E)-\sin(D-E)]/k2$

$h(4) = 1/2^{1/2}-h(0)-h(2)$

$h(5) = 1/2^{1/2}-h(1)-h(3)$,Where $k1=4x2^{1/2}$, $k2 = 2x2^{1/2}$,Here the haar coefficients are generated for any D=E and length -4 coefficients result if $E=0$ with *D* being the free parameter. The length -4 Daubechies coefficients are calculated for D=60 and E=0.Where the angles are *D, E.*Where $D = arc\ tan[(2\ h^2(0)+h^2(1)-1+(h(2)+h(3))/\ 2^{1/2}|\ 2(h(1)h(2)-h(0)h(3)+\ 2^{1/2}\ (h(0)-h(1))]$,$E = D- arc\ tan\ [(\ h(2)-h(3)\ )|(\ h(2)+h(3)-1/2^{1/2}\ )]$.

2.12 All Available Filter Coefficients

For this project bi-orthogonal 3.7 wavelet is used. The available filter banks are shown in below table [3-4].

Wavelet Families	**Wavelets**
1. Daubechies	'db1' or 'haar', 'db2', ... ,'db10', ... , 'db45'
2. Coiflets	'coif1', ... , 'coif5'
3. Symlets	'sym2', ... , 'sym8', ...,'sym45
4. Discrete Meyer	'dmey'
5. Biorthogonal	'bior1.1', 'bior1.3', 'bior1.5' 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8' 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7' 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8'
6. Reverse Biorthogonal	'rbio1.1', 'rbio1.3', 'rbio1.5' 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8' 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7' 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8'

2.13 Relationship of Filters to Wavelet Shapes

In the section "Reconstruction Filters", the importance of choosing the right filters is explained. In fact, the choice of filters not only determines whether perfect reconstruction is possible, it also determines the shape of the wavelet it uses to perform the analysis. To construct a wavelet of some practical utility, it seldom starts by drawing a waveform. Instead, it usually makes more sense to design the appropriate quadrature mirror filters, and then use them to create the waveform. By taking the low pass reconstruction filter (*L'*) for the db2 wavelet, it is seen the relation of filter to wavelet [4].

Wavelet function position

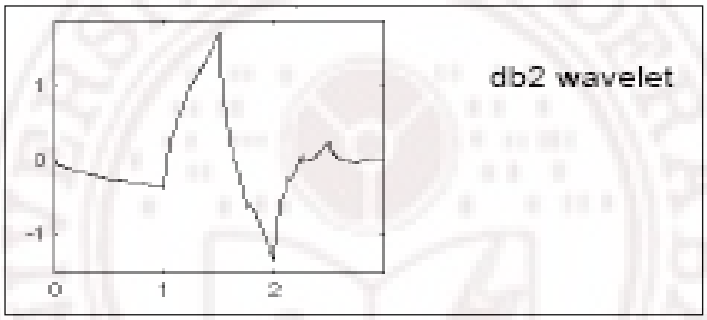


Figure 2.13a db2 wavelet

The filter coefficients can be obtained from the dbaux command:

Lprime = dbaux (2)

Lprime = 0.3415 0.5915 0.1585 –0.0915

If it reverses the order of this vector, and then multiply every even

Sample by –1, it obtains the high pass filter *H':*

Hprime = –0.0915 –0.1585 0.5915 –0.3415

Next, up sample *Hprime* by two, inserting zeros in alternate

Positions:

HU =–0.0915 0 –0.1585 0 0.5915 0 –0.3415 0

Finally, convolve the up sampled vector with the original low pass filter:

H2 = conv (*HU, Lprime*);

Plot (*H2*)

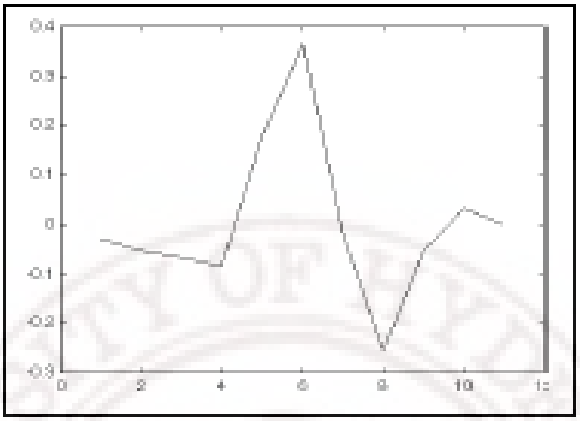


Figure 2.13b Example wavelet plot

If it iterates this process several more times, repeatedly up sampling and convolving the resultant vector with the four-element filter vector Lprime, a pattern begins to emerge.

The curve begins to look progressively more like the db2 wavelet. This means that the wavelet's shape is determined entirely by the coefficients of the reconstruction filters. This relationship has profound implications. It means that it cannot choose just any shape, call it a wavelet, and perform an analysis. At least, it can't choose an arbitrary wavelet waveform if it want to be able to reconstruct the original signal accurately. It is compelled to choose a shape determined by quadrature mirror decomposition filters.

References

[1] Introduction to Wavelet and Wavelet Transforms By C.Sidney Ramesh, A. Gopinath, Haitao Guo, Houstan, Texas, York Town, and Sunnyvale, California.

[2]. Series in Approximation and Decompositions-vol.1, Wavelet: An elementary Treatment of theory And Applications by Tom.H.KoorWinder.

[3].Dubieties, I. (1992), Ten lectures on wavelets, CBMS-NSF conference series in applied mathematics. SIAM Ed.

[4].Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," IEEE Pattern Anal. and Machine Intell. vol. 11, no. 7, pp. 674-693.

CHAPTER 3

COMPRESSION TECHNIQUES

3.1 Introduction

One of the important aspects of image storage is its efficient Compression. To make this fact clear, an example is given. An image, 1024 pixel x 1024 pixel x 24 bit without compression would require 3 MB of storage and 7 minutes for transmission, utilizing a high speed 64 Kbits/s ISDN line. If the image is compressed at a 10:1 compression ratio, the storage requirement is reduced to 300 KB and the transmission time drops to under 6 seconds. Seven 1 MB images can be compressed and transferred to a floppy disk in less time than it takes to send one of the original files, uncompressed, over an AppleTalk network.

In a distributed environment large image files remain a major bottleneck within systems. Compression is an important component of the solutions available for creating file sizes of manageable and transmittable dimensions. Increasing the bandwidth is another method, but the cost sometimes makes this a less attractive solution.

Platform portability and performance are important in the selection of the compression/decompression technique to be employed. Compression solutions today are more portable due to the change from proprietary high-end solutions to accepted and implemented international standards. JPEG is evolving as the industry standard technique for the compression of continuous tone images.

3.2 Categories of data compression algorithms

Two categories of data compression algorithm can be distinguished: lossless and lossy.Loss techniques cause image quality degradation in each compression/decompression step. Careful consideration of the human visual perception ensures that the degradation is often unrecognizable, though this depends on the selected

compression ratio. In general, lossy techniques provide far greater compression ratios than lossless techniques. Here it'll be discussed the roles of the following data compression techniques:

A. Lossless coding techniques

1. Run length encoding

2. Huffman encoding

3. Entropy coding

4. Area coding

B.Lossy coding techniques

1. Transform coding (DCT/Wavelets/Gabor)

2. Vector quantization

3. Segmentation and approximation methods

4. Spline approximation methods (Bilinear Interpolation/Regularization)

5. Fractal coding (texture synthesis, iterated functions system [IFS], recursive IFS [RIFS])

6. Efficiency and quality of different lossy compression techniques

3.3 Lossless coding techniques

Lossless coding guaranties that the decompressed image is absolutely identical to the image before compression. This is an important requirement for some application domains, e.g. medial imaging, where not only high quality is in demand, but unaltered archiving is a legal requirement. Lossless techniques can also used for the compression of other data types where loss of information is not acceptable, e.g. text documents and program executables.

Some compression methods can be made more effective by adding a 1D or 2D delta coding to the process of compression. These deltas make more effective use of run length encoding, have (statistically) higher maxima in code tables (leading to better results in Huffman and general entropy codings), and build greater equal value areas usable for area coding.

Some of these methods can easily be modified to be lossy. Lossy element fits perfectly into 1D/2D run length search. Also, logarithmic quantization may be inserted to provide better or more effective results.

3.3.1 Run length encoding

Run length encoding is a very simple method for compression of sequential data. It takes advantage of the fact that, in many data streams, consecutive single tokens are often identical. Run length coding is easily implemented, either in software or in hardware. It is fast and very well verifiable, but its compression ability is very limited.

3.3.2 Huffman encoding

This algorithm, developed by D.A. Huffman, is based on the fact that in an input stream certain tokens occur more often than others. The compression ratio achieved by Huffman encoding uncorrelated data becomes something like 1:2. On slightly correlated data, as on images, the compression rate may become much higher, the absolute maximum being defined by the size of a single input token and the size of the shortest possible output token.

3.3.3 Entropy coding

This method becomes very efficient even on virtually random data. The average compression on text and program data is about 1:2; the ratio on image data comes up to 1:8 on the average GIF image. Here again, a high level of input noise degrades the efficiency significantly. Entropy coders are a little tricky to implement, as there are usually a few tables, all growing while the algorithm runs.

3.3.4 Area coding

Area coding is an enhanced form of run length coding, reflecting the two-dimensional character of images. This is a significant advance over the other lossless methods. The possible performance of this coding method is limited mostly by the very high complexity of the task of finding largest areas with the same characteristics.. This type of coding can be highly effective but it bears the problem of a nonlinear method, which cannot be implemented in hardware.

3.4 Lossy coding techniques

In more applications, it has no need in the exact restoration of stored image. This fact can help to make the storage more effective, and this way it gets to lossy compression methods. Lossy image coding techniques normally have three components: Image modeling which defines such things as the transformation to be applied to the image. Parameter quantization where by the data generated by the transformation is quantized to reduce the amount of information .Encoding, where a code is generated by associating appropriate codeword to the raw data produced by the quantization.

Each of these operations is in some part responsible of the compression. Image modeling is aimed at the exploitation of statistical characteristics of the image (i.e. high correlation, redundancy). Typical examples are transform-coding methods, in which the data is represented in a different domain (for example, frequency in the case of the Fourier Transform [FT], the Discrete Cosine Transform [DCT], the Kahrunen-Loewe Transform [KLT], and so on), where a reduced number of coefficients contain most of the original information. In many cases this first phase does not result in any loss of information.

The aim of quantization is to reduce the amount of data used to represent the information within the new domain. Quantization is in most cases not a reversible operation: therefore, it belongs to the so-called 'lossy' methods.

Encoding is usually error free. It optimizes the representation of the information (helping sometimes to further reduce the bit rate), and may introduce some error detection codes.

In the following sections, a review of the most important coding schemes for lossy compression is provided. Some methods are described in their canonical form (transform coding, region based approximations, fractal coding, wavelets, hybrid methods) and some variations and improvements presented in the scientific literature are reported and discussed.

3.4.1 Transform coding (DCT/Wavelets/Gabor)

A general transform coding scheme involves subdividing an NxN image into smaller nxn blocks and performing a unitary transform on each sub image. A unitary transform is a reversible linear transform whose kernel describes a set of complete, ortho normal discrete basic functions. The goal of the transform is to decorate the original signal, and this declaration generally results in the signal energy being redistributed among only a small set of transform coefficients. In this way, many coefficients may be discarded after quantization and prior to encoding. Transform coding can be generalized into four stages [1]:

1. Image subdivision

2 Image transformations

3. Coefficient quantization

4. Encoding.

For a transform coding scheme, logical modeling is done in two steps: a segmentation one, in which the image is subdivided in bi dimensional vectors (possibly of different sizes) and a transformation step, in which the chosen transform (e.g. KLT, DCT, and Hadamard) is applied.

Quantization can be performed in several ways. Most classical approaches use 'zonal coding', consisting in the scalar quantization of the coefficients belonging to a predefined area , and 'threshold coding', consisting in the choice of the coefficients of each block characterized by an absolute value exceeding a predefined threshold.

3.4.2 Vector quantization

A vector quantization can be defined mathematically as a transform operator T from a K-dimensional Euclidean space R^K to a finite subset X in R^K made up of N vectors. The algorithm is derived from the KNN cauterization method, and is performed by iterating the following basic operations: 1) Subdivide the training set into N groups (called 'partitions' or 'Voronoi regions'), which are associated with the N codebook letters, according to a minimum distance criterion. 2) The centroids of the Voronoi regions become the updated codebook vectors.3) compute the average distortion: if the percent reduction in the distortion (as compared with the previous step) is below a certain threshold, then stop.

3.4.3 Segmentation and approximation methods

With segmentation and approximation coding methods, the image is modeled as a mosaic of regions, each one characterized by a sufficient degree of uniformity of its pixels with respect to a certain feature (e.g. grey level, texture); each region then has some parameters related to the characterizing feature associated with it. The operations of finding a suitable segmentation and an optimum set of approximating parameters are highly correlated, since the segmentation algorithm must take into account the error produced by the region reconstruction . These two operations constitute the logical modeling for this class of coding schemes; quantization and encoding are strongly dependent on the statistical characteristics of the parameters of this approximation.

3.5 Efficiency and quality of different lossy compression techniques

The performances of lossy picture coding algorithms are usually evaluated on the basis of two parameters:

a. The compression factor (or analogously the bit rate) and

b. The distortion produced on the reconstruction.

The first is an objective parameter, while the second strongly depends on the usage of the coded image. Nevertheless, considering an objective measure of the error, like MSE or SNR, can make a rough evaluation of the performances of a method.

3.6 Comparison of Different Compression Methods

Some standardization processes based on transform coding, such as JPEG, have been started. Performances of such a standard are quite good if compression factors are maintained under a given threshold. Over this threshold, artifacts become visible in the reconstruction and tile effect affects seriously the images decoded, due to quantization effects of the DCT coefficients.

On the other hand, there are two advantages: first, it is a standard, and second, dedicated hardware implementations exist. For applications which require higher compression factors with some minor loss of accuracy when compared with JPEG, different techniques should be selected such as wavelets coding or spline interpolation, followed by an efficient entropy encoder such as Huffman, arithmetic coding or vector quantization.

Some of these coding schemes are suitable for progressive reconstruction (Pyramidal Wavelet Coding, Two Source Decomposition, etc). This behavior can be exploited by applications such as coding of images in a database, for previewing purposes or for transmission on a limited bandwidth channel.

3.7 Lossy compression Vs Lossless compression

The advantage of lossy methods over lossless methods is that in some cases a lossy method can produce a much smaller compressed file than any known lossless method, while still , it is meeting the requirements of the application.

Lossy methods are most often used for compressing sound, images or videos. The compression ratio of lossy video codec's is nearly always far superior to those of the audio and still-image equivalents. Audio can be compress at 1:10 with no noticeable loss of the quality; video can be compressed immensely with little visible quality loss, e300:1. Lossy compressed images are still compressed to 1/10t their original size, as with audio, but the quality loss is more noticeable, especially on closer inspection. When a user acquires a lossy-compressed file, the retrieved file can be quite different to the original at the bit level while being indistinguishable to the human ear or eye for most practical purposes. The human eye can see only certain frequencies of light. The psychoacoustic model describes how sound can be highly compressed without degrading the perceived quality of the sound.

3.8 Image Coding Using DWT

3.8.1 One-Stage Filtering: Approximations and Details

For many signals, the low-frequency content is the most important part. It gives the signal its identity. The high-frequency content on the other hand imparts flavor or nuance. In human voice, if it removes the high-frequency components, the voice sounds different but it can still tell what's being said. However, if it removes enough of the low-frequency components, it hears gibberish. In wavelet analysis, it often speaks of approximations and details. The approximations are the high-scale, low-frequency components of the signal. The details are the low-scale, high-frequency components.
The filtering process at its most basic level looks like this shown in 3.8.1a:

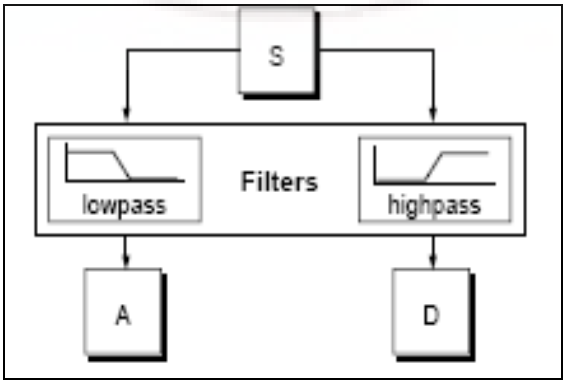


Figure 3.8.1a Basic Level Filtering Process

The original signal S passes through two complementary filters and emerges as two signals [2]. If it actually performs this operation on a real digital signal, it winds up with twice as much data as it started with. Suppose, for instance that the original signal S consists of 1000 samples of data. Then the resulting signals will each have 1000 samples, for a total of 2000.

These, A and D are shown in figure 3.8.1b, but it gets 2000 values instead of the 1000, it had. There exists a more subtle way to perform the decomposition using wavelets. By looking carefully at the computation, it may keep only one point out of two in each of the two 2000-length samples to get the complete information. This is the notion of own sampling. It produces two sequences called cA and cD.

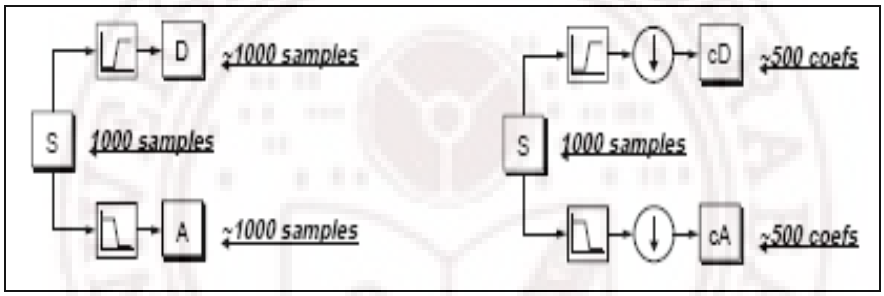


Figure 3.8.1b Samples Generation

The process on the right, which includes down sampling, produces DWT Coefficients. To gain a better appreciation of this process let's perform a one-stage discrete wavelet transform of a signal. Our signal will be a pure sinusoid with high-frequency noise added to it. Here is schematic diagram, figure 3.8.1c with real signals inserted into it:

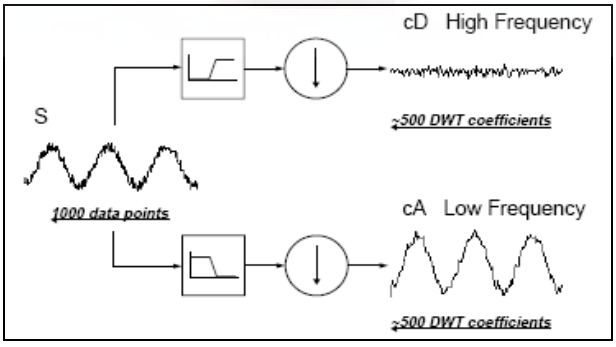


Figure 3.8.1c Approximate and Detail Coefficients

It may observe that the actual lengths of the detail and approximation coefficient vectors are slightly more than half the length of the original signal. This has to do with the filtering process, which is implemented by convolving the signal with a filter. The convolution smears the signal, introducing several extra samples into the result.

3.8.2 Multiple-Level Decomposition

The decomposition process can be iterated, with successive approximations being decomposed in turn, so that one signal is broken down into many lower resolution components. This is called the wavelet decomposition tree shown in 3.8.2a.

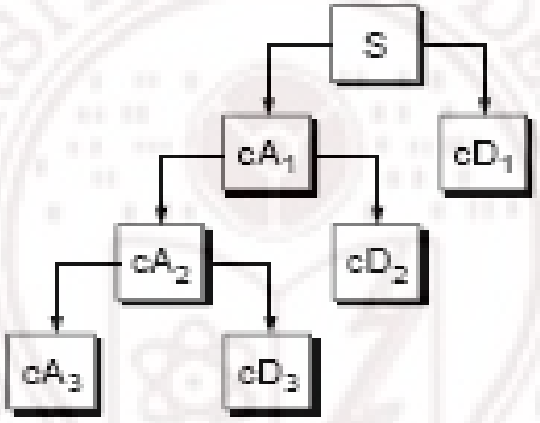


Figure 3.8.2 Multilevel Decomposition

.

3.8.3 Number of Levels

Since the analysis process is iterative, in theory it can be continued indefinitely. In reality, the decomposition can proceed only until the individual details consist of a single sample or pixel. In practice, it'll select a suitable number of levels based on the nature of the signal, or on a suitable criterion such as entropy.

3.8.4 Wavelet Reconstruction

It is learned how the discrete wavelet transform can be used to analyze or decompose signals. This process is called decomposition or analysis. The other half of

the process is how those components can be assembled back into the original signal without loss of information. This process is called reconstruction, or synthesis, shown in figure 3.8.4a. The mathematical manipulation that effects synthesis is called the inverse discrete wavelet transforms (IDWT).

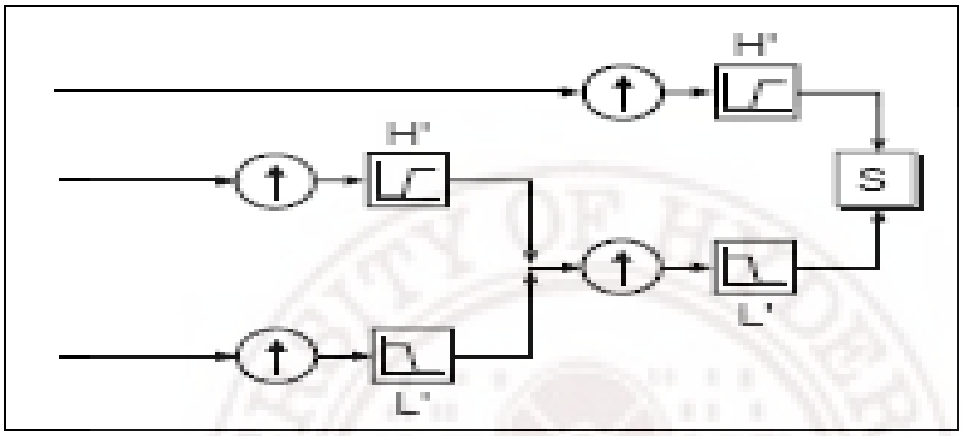


Figure 3.8.4a Wavelet Reconstruction

Where wavelet analysis involves filtering and down sampling, the wavelet reconstruction process consists of up sampling and filtering. Up sampling is the process of lengthening a signal component by inserting zeros between samples is shown in figure 3.8.4b:

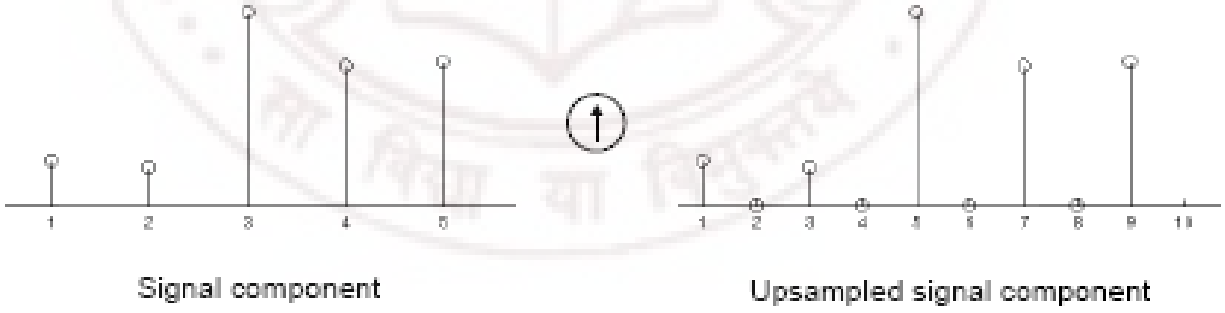


Figure 3.8.4b Up sampling process

3.8.5 Reconstruction Filters

The down sampling of the signal components performed during the decomposition phase introduces a distortion called aliasing. By carefully choosing filters for the decomposition and reconstruction phases that are closely related (but not identical), it can "cancel out" the effects of aliasing [3].

The low- and high pass decomposition filters (L and H), together with their associated reconstruction filters (L' and H'), form a system of what is called quadrature mirror filters is shown in figure 3.8.5:

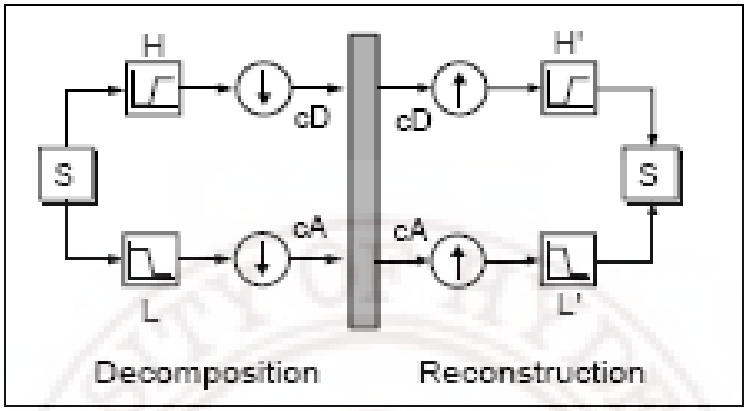


Figure3. 8.5 Reconstruction Filter

3.8.6 Reconstructing Approximations and Details

It is seen that it is possible to reconstruct our original signal from the coefficients of the approximations and details.

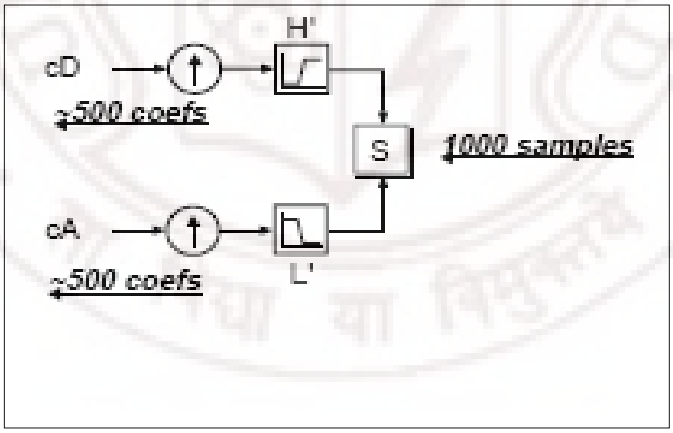


Figure 3.8.6a Reconstructing Approximation and Detail coefficients

It is also possible to reconstruct the approximations and details themselves from their coefficient vectors.

As an example is given that how it would reconstruct the first-level approximation A1 from the coefficient vector cA1 in figures 3.8.6. It passes the coefficient vector cA1 through the same process it used to reconstruct the original signal. However, instead of

combining it with the level-one detail cD1, it feeds in a vector of zeros in place of the detail coefficients vector:

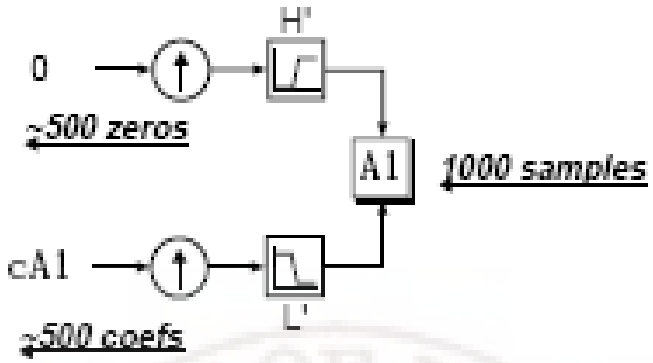


Figure 3.8.6b Reconstructing Approximation coefficients

The process yields a reconstructed approximation A1, which has the same length as the original signal S and which is a real approximation of it. Similarly, it can reconstruct the first-level detail D1, using the analogous process:

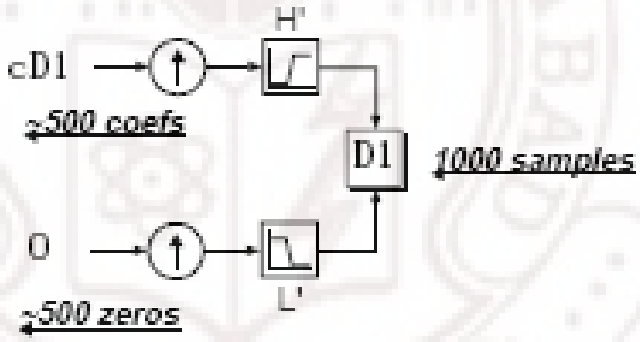


Figure 3.8.6c Reconstructing Detail coefficients

The reconstructed details and approximations are true constituents of the original signal. In fact, it finds when it combines them that:

$$A1 + D1 = S$$

Note that the coefficient vectors cA1 and cD1—because they were produced by down sampling and are only half the length of the original signal — cannot directly be combined to reproduce the signal.

It is necessary to reconstruct the approximations and details before combining them [4]. Extending this technique to the components of a multilevel analysis, it finds that similar relationships hold for all the reconstructed signal constituents.

There are several ways to reassemble the original signal:

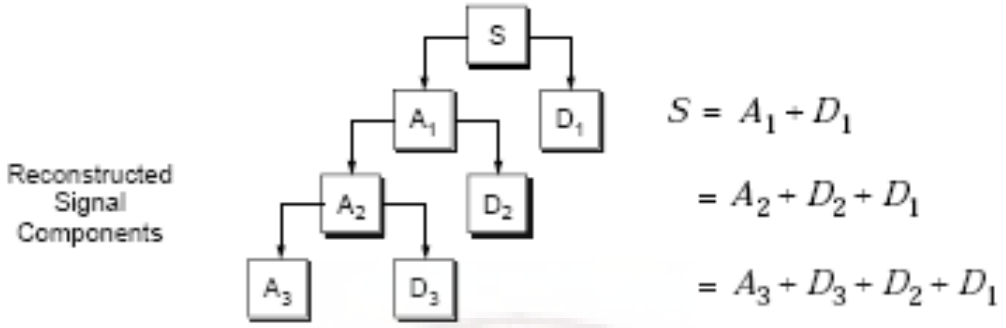


Figure 3.8.6d Reconstructing process

3.8.7 Multi-step Decomposition and Reconstruction

A multi step analysis-synthesis process can be represented as, shown in figure 3.8.9:

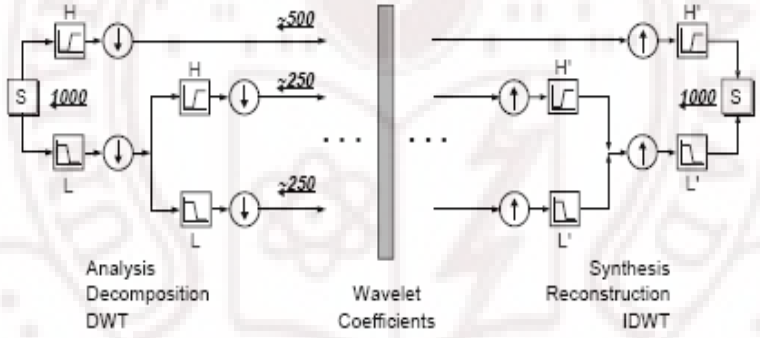


Figure 3.8.7 Multi-step Decomposition and Reconstruction

This process involves two aspects: breaking up a signal to obtain the wavelet coefficients, and reassembling the signal from the coefficients. It has already discussed decomposition and reconstruction at some length. There is no point breaking up a signal merely to have the satisfaction of immediately reconstructing it. It modifies the wavelet coefficients before performing the reconstruction step using threshold step.In this work used threshold type is hard type, thresholding and it's types are explained in section 3.8.9.

3.8.8 The Scaling Function

It is seen the interrelation of wavelets and quadrature mirror filters. The wavelet function ψ is determined by the high pass filter, which also produces the details of the wavelet decomposition.

The scaling function is very similar to the wavelet function. It is determined by the low pass quadrature mirror filters, and thus is associated with the approximations of the wavelet decomposition. In the same way that iteratively up- sampling and convolving the high pass filter produces a shape approximating the wavelet function, iteratively up-sampling and convolving the low pass filter produces a shape approximating the scaling function.

3.8.9Thresholding Methods Before reconstruction process, except the approximation values, the remaining horizontal, vertical, diagonal coefficients will be thresholded, and then they are sent to the reconstruction process.

There are two thresholding methods. They are

A. Hard thresholding.

B. Soft thresholding.

A. Hard thresholding: Hard thresholding is the usual process of setting to zero the coefficients whose absolute values are lower than specific threshold value. That means, the values of the detailed coefficients less than the threshold value are made to zero. The hard threshold method will produce discontinuities at threshold value. The Hard thresholding estimator can be expressed as

$$x_{hard}(t) = \begin{cases} x(t) & \text{if} |x(t)| > TH \\ 0 & \text{if} |x(t)| \le TH \end{cases} \quad \text{…….. (3.8.9a)}$$

B. Soft thresholding: Soft thresholding is an extension of hard thresholding by first setting to zero coefficients whose absolute values are lower than the specific threshold and then shrinking the nonzero coefficients toward zero[5,6]. Since soft thresholding

thresholds values uniformly, we get a reconstructed signal with smaller mean square error. The Soft thresholding estimator can be expressed as

$$x_{soft}(t) = \begin{cases} \text{sign}(x(t))(|x(t)| - TH) & \text{if } |x(t)| > TH \\ 0 & \text{if } |x(t)| \le TH \end{cases}$$

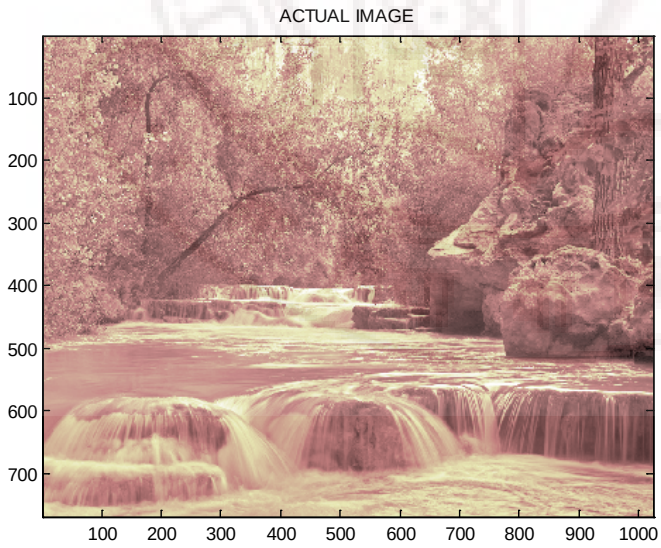
…… (3.8.9b)

The complete explanation of architecture used in matlab and vhdl is explained in section 4.3.

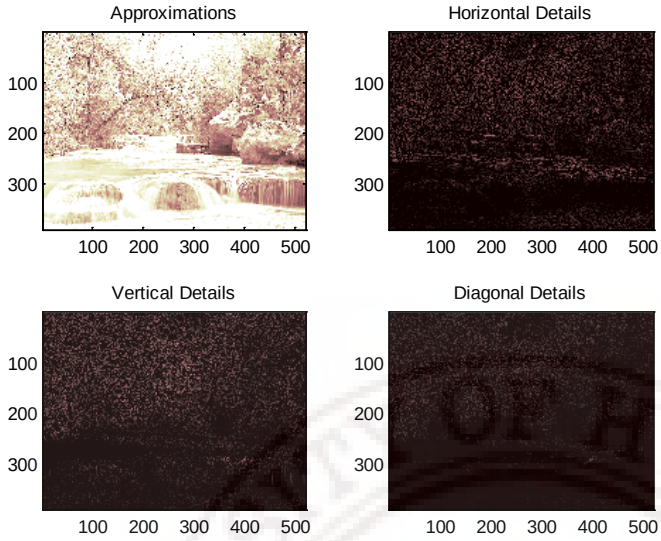
3.9 Simulation Results in MATLAB

Explanation about Results: The figure is shown in section A is the in put image and is given to the image coding system using wavelets at level 2 decomposition,bior-3.7as the filter bank, hard threshold value selected is 7.4129.Then the output image is shown in the section c. The input image size is 179k bytes, the resulting output image size 156k bytes. The achieved compression ratio is 1.1474. In the section B, the figures of approximate, detail, vertical, diagonal coefficients are shown.

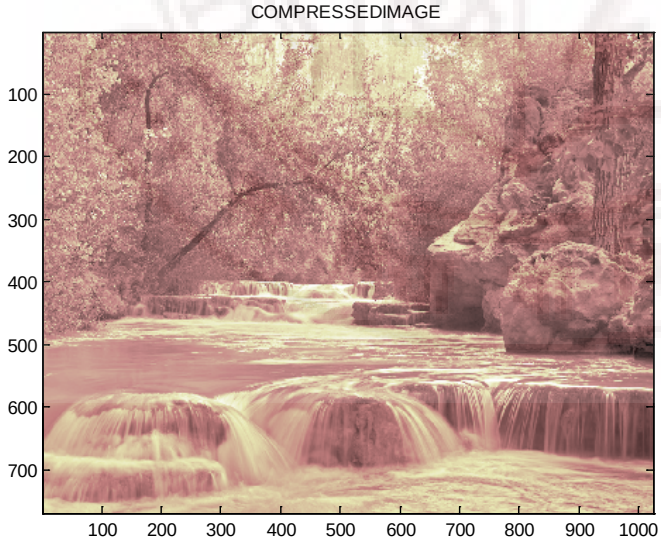
A. Input image (179KB)



B. Figures of coefficients



C. Output Image In MATALB (156KB)



Compression Ratio = 179/156 = 1.1474

Threshold = 7.4129, Threshold Type: hard

Number of levels =2

Filter coeff.Used: Bi-orthogonal 3.7

.

References

[1]. Digital Image Processing using MATLAB--Rafael C. Gonzalez, Richard E. Woods Steven C. Eddies [1]

Digital Image Processing using MATLAB--Rafael C. Gonzalez, Richard E. Woods Steven C. Eddies, Guttmann, Peter (1995) -- Introduction to Data Compression.

Lane, Tom (1995)--Introduction to JPEG Compression

[2]. DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," IEEE Trans. on Inf. Theory, vol. 38, No 2, pp. 719-746.

[3]. Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in Progress in wavelet analysis and applications, Y. Meyer, S. Roques, pp. 109-128. Frontières Ed.

[4].Series in Approximation and Decompositions-vol.1, Wavelet: An elementary Treatment of theory And Applications by Tom.H.KoorWinder

[5] M.Kania**,** M.Fereniec**,** R.Maniewski Wavelet Denoising for Multi-lead High Resolution ECG signals, MEASUREMENT SCIENCE REVIEW, volume 7, section 2, No.4, 2007.

[6] Donoho D.L: De-noising by Soft-thresholding. *IEEE Trans.* On Inf. Theory, 1995, Vol.41, 3, pp 613-627.

Mallat S, *A* Theory for Multiresoltion signal decomposition: the wavelet representation, IEEE Trans.on IT. 1992, 38(2), pp 674-693.

CHAPTER 4

HARDWARE RELIZATION

4.1 FPGA Design Flow

The FPGA Design flow is shown in following figure 4.1. The standard design flow comprises the following steps.

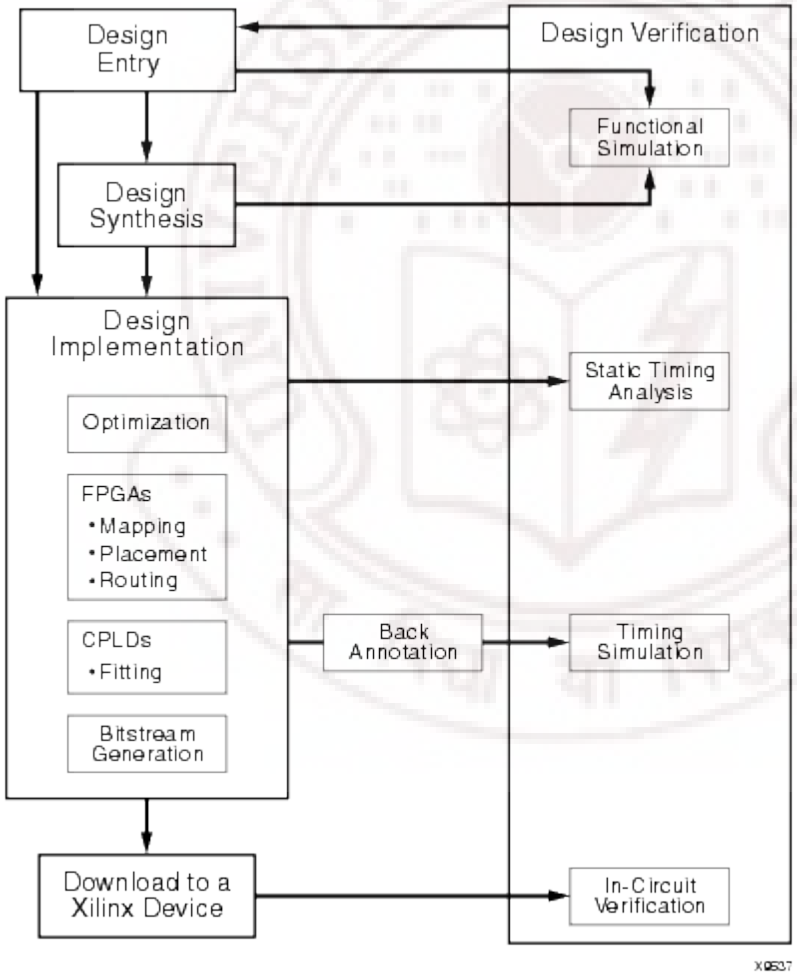


Figure 4.1 FPGA Design Flow

Design Entry and Synthesis—In this step of the design flow, the design is created using a Xilinx-supported schematic editor, a hardware description language (HDL) for text-based entry, or both. If it uses an HDL for text-based entry, it must synthesize the HDL file into an EDIF file or, if it is using the Xilinx Synthesis Technology (XST) GUI, it must synthesize the HDL file into an NGC file [1].

Design Implementation—By implementing to a specific Xilinx architecture, it converts the logical design file format, such as EDIF, that it created in the design entry and synthesis stage into a physical file format. The physical information is contained in the native circuit description (NCD) file for FPGAs and the VM6 file for CPLDs. Then it creates a bitstream file from these files

Design Verification—Using a gate-level simulator or cable, it is ensured that the design meets it's timing requirements and functions properly. The full design flow is an iterative process of entering, implementing, and verifyingy the design until it is correct and complete.

4.2 Tools Used

For this project, the used tools are MATLAB, XILINX ISE 9.1i, TURBO C. MATLAB is a high-level language and interactive environment that enables us to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran. The complete task about this project is done using MATLAB version 7.2 and many images are given as inputs and outputs are obtained. Xilinx is a developer of FPGA and CPLD devices that are used in numerous applications within telecommunications, automotive, consumer, defense, and other fields. Here in this case it is being used for image processing and coding purpose. For this taskField Programmable Gate Array is chosen and for this reason explained in the introduction part. TURBO C is being used to varify the results and to check the intermediate results

4.3 Architecture

The figure given below explains the decomposition and reconstruction process step by step.

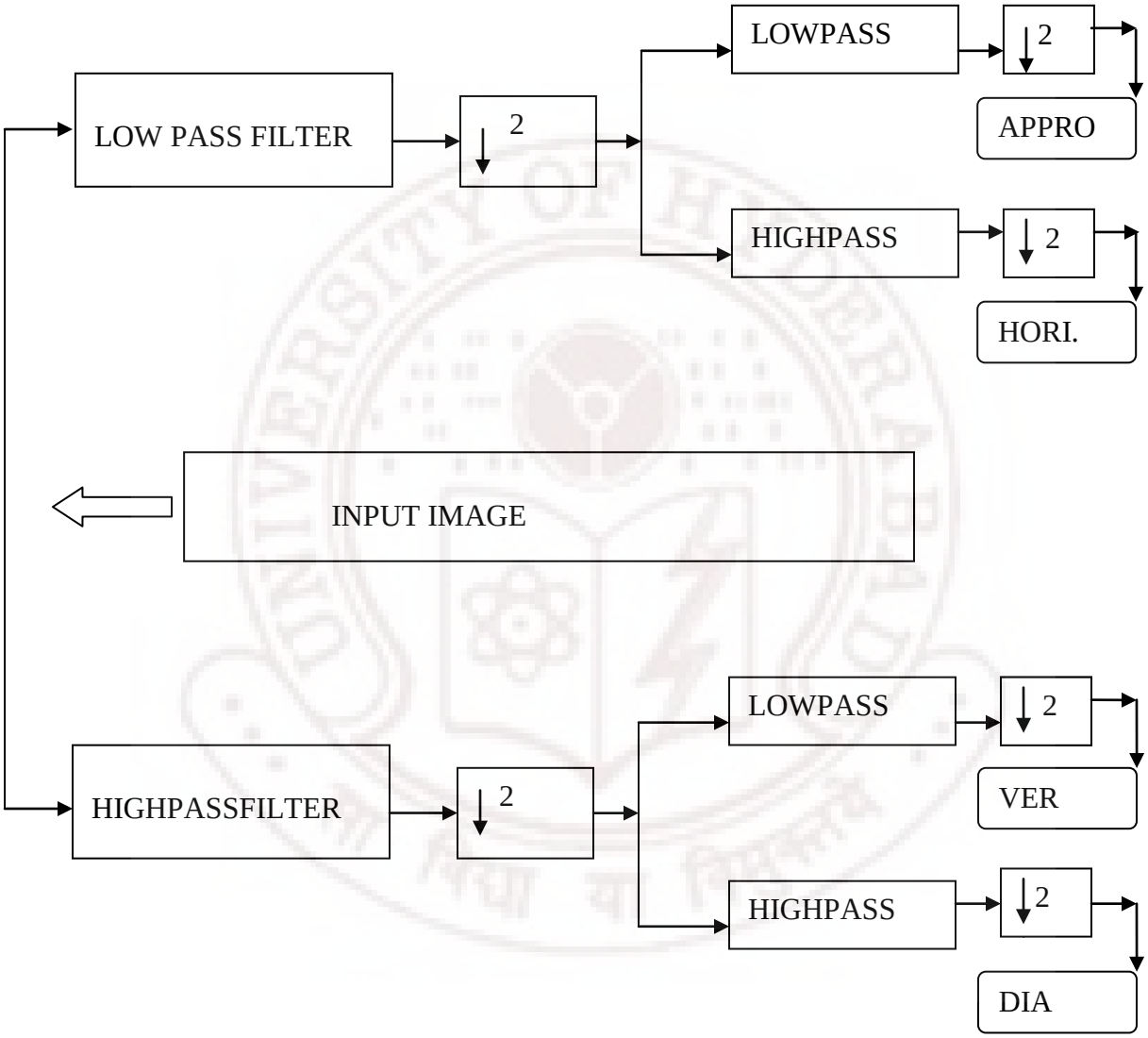


Figure 4.3a Decomposition Architecture

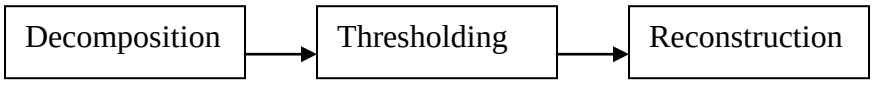


Figure 4.3b Steps in Image Coding

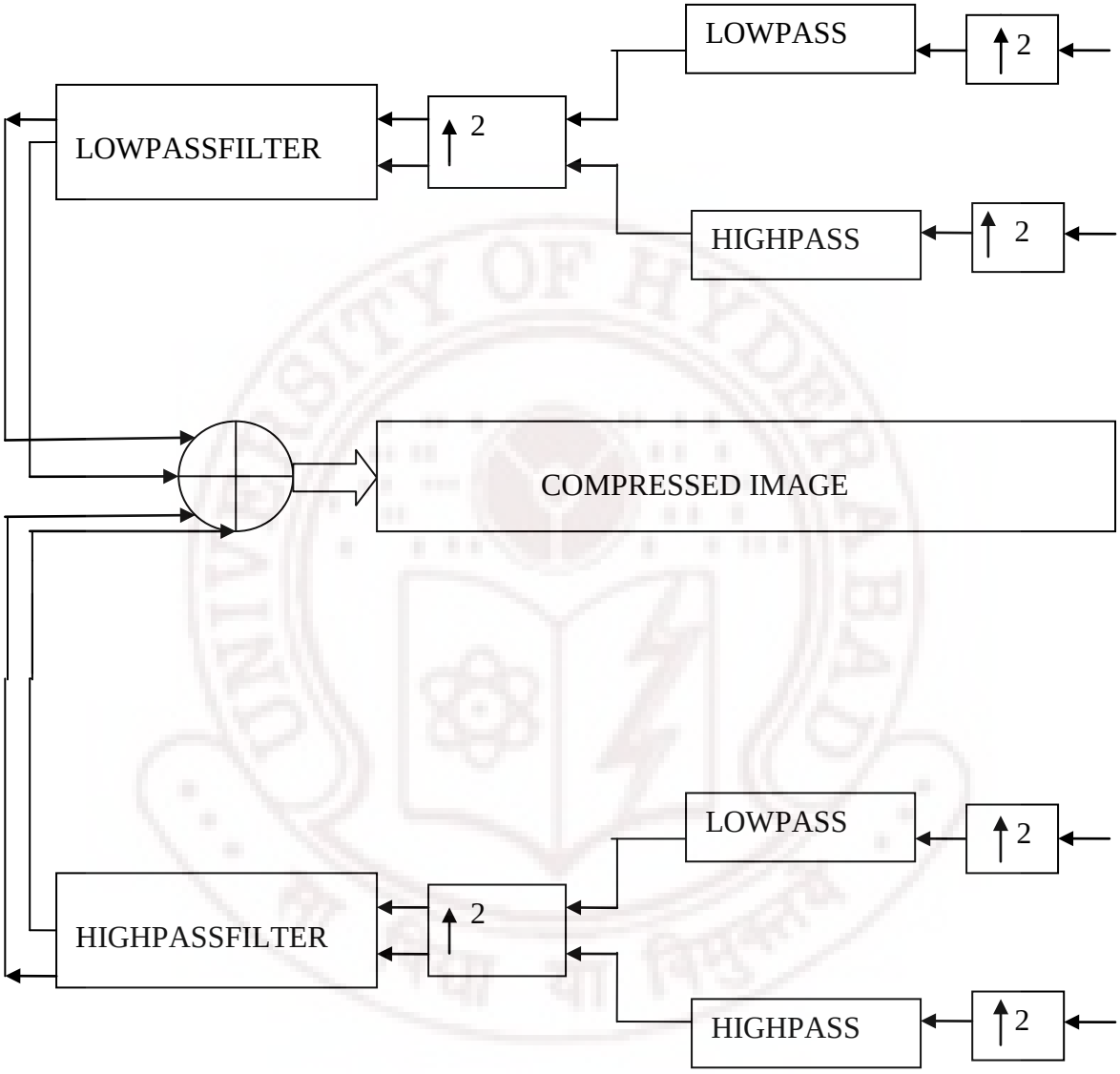


Figure 4.3c Reconstruction Architecture

For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets. The above figure describes the basic decomposition and

reconstruction step for images. In first step an input image is fed to the cascaded filter section shown in above figure 4.3a. This two dimensional data contains horizontal and vertical values. The intensities are fed to the first filter as row –wise. Here basic operation of the first filter is row-wise convolution. And filter coefficients are added to the sufficient number of zeros to avoid unknown values in convolution process, which is basic operation of a filter.

The intermediate values generated are next fed to the second filter in cascaded to the first filter as column-wise. Before this these are down sampled by the factor of two. Here basic operation is column-wise operation. Down sampling operation is performed on the result generated here by two.

If the cascaded filter section is Low-Low, the coefficients are called "approximation" and if Low High, High-Low, High-High, they are called "horizontal, "vertical", "diagonal". Except the approximation coefficients remaining are thresholded by the appropriate threshold, because they contain important information about the input image. The value which is less than threshold will be made to zero.

In the next step, reconstruction shown in figure 4.3c, the inverse operation is performed. Cascaded filter section is converted in to the Low-Low, High-Low, Low-High, and High-High for the approximations and thresholded remaining coefficients respectively. Instead of down sampling, up sampling is performed.

Finally the generated coefficients are summed up and central part of the summed result is taken to give the compressed two-dimensional data.

4.4 Simulation

Converting In To Image**:** The input given to the design in vhdl is first of all, converted into the binary form and the filter coefficients are taken in the form of 2s complement representation. In convolution, multiplicand (A) contains some x-bit number of integer part and y-bit number of fractional part. Similarly, multiplier (B) contains some p-bit number of integer part and q-bit number of fractional part. Then the multiplication values of these two real values give the resulting binary domain as A*B, contains x+y+p+q number of bits in it. In this result, (x + y) number of bit s gives the integer part of real result and the (p +q) number of bits will give the fractional part

The output of the simulation of VHDL is in 2scomplemnt form where x+y bits indicate integer part, and p+q indicates fractional part. These binary values are converted in to the decimal values and output will be seen using matlab.

4.5 Simulation Results

The final results are compared got from vhdl with matlab shown clearly in below table**.**

IMAGE	ACTUAL SIZE	NUMBER OF COEFFICIENTS (MATALB)	NUMBER OF COEFFICIENTS (VHDL)	MATLAB RESULT	VHDL RESULT	THR USED
13BY12	**720B**	**156**	**156**	**714B**	**707B**	**35**
18BY20	**852B**	**360**	**360**	**816B**	**816B**	**90**

Compression Ratio = 720/707 = 1.0184 for 13by12 input image in VHDL.

Compression Ratio = 720/714 = 1.0084 for 13by12 input image in MATLAB.

Here the same input (13by12) is given to the vhdl image compression system at level one decomposition, bior3.7 bank filter, threshold selected is 35. The achieved compression ratio is 1.0184 which is slightly greater than what achieved in matlab (1.0084).

4.6 Synthesis Report

The complete functional, synthesizable, implementable code is split in to two parts because the complete code is not fit in vertex-5. The two parts are, first one is 1.Decomposition,Thresholding, second is 2.reconstruction. Their synthesis, timing reports are given in below tables.

The synthesis, map and Place and Route reports for VERTEX4 FPGA for Selected Device: 4vlx15sf363-12 from Xilinx ISE 9.1 is given below the synthesis, map and Place and Route reports from Xilinx ISE 9.1 are given below.

Device utilization summary:

Device utilization summary is given below clearly in given table. And Selected Device is 4vlx15sf363-12

Utilized Device	Number
Number of Slices	131 out of 6144
Number of Slice Flip Flops	145 out of 12288
Number of 4 input LUTs	248 out of 12288
Number of IOs	270
Number of bonded IOBs	197 out of 240
Number of GCLKs	1 out of 32

Partition Resource Summary: No Partitions were found in this design.

Asynchronous Control Signals Information: No asynchronous control signals found in this design

Timing Summary

Speed Grade	-12
Minimum period	Minimum period: 5.122ns(195.229MHz)
Minimum input arrival time before clock	No path found
Maximum output required time after clock	4.130ns
Maximum combinational path delay	No path found

2.

Device utilization summary

Selected Device	4vlx15sf363-12
Number of Slices	21673 out of 6144
Number of Slice Flip Flops	138 out of 12288
Number of 4 input LUTs	41951 out of 12288
Number of IOs	238
Number of bonded IOBs	238 out of 240
Number of GCLKs	1 out of 32
Number of DSP48s	32 out of 32
Partition Resource Summary	No Partitions were found in this design

TIMING REPORT

Asynchronous Control Signals Information: No asynchronous control signals found in this design

Timing Summary:

Speed Grade	-12
Minimum period	5.738ns(MaximumFrequency:174.266MHz)
Minimum input arrival time before clock	33.032ns
Maximum output required time after clk	3.793ns
Maximum combinational path delay	No path found

4.7 Map Report

After mapping the design summary is given below.

1.

Number of errors	0
Number of warnings	2
Number of Slice Flip Flops	86 out of 12,288
Number of 4 input LUTs	186 out of 12,288
Number of occupied Slices	126 out of 6,144
Number of Slices containing only related	126 out of 126
Number of Slices containing unrelated	0 out of 126
Total Number of 4 input LUTs	248 out of 12,288
Number used as logic	186
Number used as a route-thru	62
Number of bonded IOBs	197 out of 240
Number of BUFG/BUFGCTRLs	1 out of 32
Total equivalent gate count for design	2,756
Additional JTAG gate count for IOBs	9,456
Peak Memory Usage	201MB
Total REAL time to MAP completion	49 secs
Total CPU time to MAP completion	23secs
Target Device	xc4vlx15
Target Package	Sf363
Target Speed	-12
Number used as logic	186

2.

Number of errors	0
Number of warnings	0
Number of Slice Flip Flops	65 out of 12,288
Number of 4 input LUTs	40,148 out of 12,288
Number of occupied Slices	21,495 out of 6,144
Number of 4 input LUTs	42,113 out of 12,288
Number of occupied Slices	21,495 out of 6,144
Number of Slices containing only	20,850 out of 21,495
Number of Slices containing	645 out of 21,495

Total Number of 4 input LUTs	42,113 out of 12,288
Number used as logic	40,148
Number used as a route-thru	1,965
Number of bonded IOBs	238 out of 240
Peak Memory Usage	524MB
Total REAL time to MAP completion	1 out of 32
Total CPU time to MAP completion	8 mins 23 secs
Number of bonded IOBs	238 out of 240
Number of BUFG/BUFGCTRLs	1 out of 32
Number used as BUFGs	1
Number of DSP48s	32 out of 32
Total equivalent gate count for design	482,985
Additional JTAG gate count for IOBs	11,424

4.8 PAR Report

All signals are completely routed.

Total REAL time to PAR completion	29 secs
Total CPU time to PAR completion	19secs
Peak Memory Usage	231MB
Placement: Completed	Completed - No errors found
Routing	Completed - No errors found
Number of error messages	0
Number of warning messages	0
Number of info messages	1
PAR done	Completed ,no Errors found

4.9 Applications

Applications of image processing are discussed below.

1) Photography and printing

2) Satellite image processing

3) Medical image processing

4) Face detection, feature detection, face identification

5) Microscope image processing

4.9.1 Photograph

Photography is the process of making pictures by means of the action of light. It involves recording light patterns, as reflected from objects, onto a sensitive medium through a timed exposure [2]. The process is done through mechanical, chemical or digital devices commonly known as cameras.

Photography can be classified under imaging technology and has gained the interest of scientists and artists from its inception. Scientists have used its capacity to make accurate recordings, such as Edward Muybridge in his study of human and animal locomotion.

Artists have been equally interested by this aspect but have also tried to explore other avenues than the photomechanical representation of reality, such as the pictorials movement. Military, police and security forces use photography for surveillance, recognition and data storage.

4.9.2 Medical imaging

Medical imaging is the process by which physicians evaluate an area of the subject's body that is not externally visible. Medical imaging may be clinically motivated, seeking to diagnose and examine disease in specific human patients. Alternatively, researchers may use it in order to understand processes in living organisms. Many of the techniques developed for medical imaging also have scientific and industrial applications.

4.9.3 Microscope image processing

Microscope image processing is a broad term that covers the use of digital image processing techniques to process, analyze and present images obtained from a

microscope. Such processing is now commonplace in a number of diverse fields such as medicine, biological research, cancer research, drug testing, metallurgy, etc. A number of manufacturers of microscopes now specifically design in features that allow interface to an image processing system.

References

[1]. Field Programmable Gate Array Design Flow, ISE help, www.xilinx.com

[2]. Image Processing and Analysis - Variation, PDE, Wavelet, and Stochastic Methods, by Tony F. Chan and Jackie (Jianhong) Shen, John C. Russ (2006). The Image Processing Handbook

CHAPTER 5

CONCLUSION

Conclusion:

By increasing threshold value it can achieve more compression ratio but with less quality. And it achieves more quality if it is done at lower thresholding values, so, a Tradeoff between these, a certain thresholding is performed at optimized threshold value for obtaining better quality and coding results. The two dimensional wavelet transform is used to get the compressed image and used filter bank is bi-orthogonal 3.7. The number of levels used is one in this work. The compression ratio changes according to the number of levels and the optimized threshold selected.

Future Work:

After successful synthesis, implementing, the bit file generated is programmed in to the selected device in FPGA to make the design hardware integrated chip .Here, in this work, The complete functional, synthesizable, implementable code is fit in the FPGA device XC5VLX30, family vertex 5, package ff324, spped-2 with a warning used 30 DSP48ES (i.e. 101%) out of available 28 in the selected device. The code is optimized and will be programmed in to the device having sufficient number of resources in it.

PROGRAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;


entity compr is

port( clk :in std_logic;

re_out_row :out std_logic_vector(72 downto 0)
);
end compr;




architecture Behavioral of compr is

-----DECLARATION OF TYPES----------------------------------------------

type xintype is array( 1 to 741) of std_logic_vector(8 downto 0);--9 bits for x values;

type xtype is array(1 to 12) of std_logic_vector(8 downto 0);--9 bits for x values;
type tdtype is array(1 to 13) of xtype ;

type xint is array(1 to 12) of integer;
type tdint is array(1 to 13) of xint ;

type htype is array(1 to 59) of std_logic_vector(15 downto 0);--h length is 16 + 31
zeros+57+2

type ctype is array(1 to 57) of std_logic_vector(24 downto 0);
type convtype is array(1 to 13) of ctype ;

type subtyp is array(1 to 13) of std_logic_vector(24 downto 0);--13 elements
type subsam is array(1 to 13) of subtyp;---13 by 13;

type tdextrtype is array(1 to 13) of std_logic_vector(24 downto 0);--y2 is 43 by13
type exttr      is array(1 to 43) of tdextrtype;
```

```
-----------------------------------------
type tdextype is array(1 to 57) of std_logic_vector(8 downto 0);--actuall
15+12+15=42,but iam taking extra columns to add zero at end
type exttc     is array(1 to 13) of tdextype ;                  --it is better for convolution ,no
need add zero at the end of "column extended row"
type convseltype is array(1 to 27)of std_logic_vector(24 downto 0);
type conv_sel is array(1 to 13) of convseltype;

type transtype is array(1 to 58) of  std_logic_vector(24 downto 0); ---------actual no of
rows 43 ---to add zeros taking extra
type sc_sel_subrext_transtype is array (1 to 13) of transtype;  ----13by(43+15columns
zeros) i.e y2dash

type sc_2_type is array ( 1 to 58) of std_logic_vector(40 downto 0);----13by58
type sc_2_typpee is array(1 to 13) of sc_2_type ;

type sc_2_trans_type is array (1 to 13) of std_logic_vector(40 downto 0);--58by13
type sc_2_trans_typpee is array(1 to 58) of sc_2_trans_type;

type sc_2_trans_selt is array(1 to 13) of std_logic_vector(40 downto 0);--28by13
type sc_2_trans_sel_type is array(1 to 28) of sc_2_trans_selt ;


type appxtype is array (1 to 13) of std_logic_vector(40 downto 0);--14by13
type appx_type is array(1 to 14) of appxtype;

---------------------------------------------------------------

TYPE AHVDTYPE IS ARRAY(1 TO 182) OF std_logic_vector(40 downto 0);
TYPE AHVDTYPEE IS ARRAY(1 TO 4) OF AHVDTYPE;


TYPE AHVD_V2TYPE IS ARRAY(1 TO 728) OF std_logic_vector(40 downto 0);


---------------------------------------------------------------

TYPE coeftype IS ARRAY(1 TO 728) OF std_logic_vector(40 downto 0);

TYPE coefappxtype IS ARRAY(1 TO 13) of std_logic_vector(40 downto 0);
TYPE coefupstype IS ARRAY(1 TO 27)OF coefappxtype;


---------------------------------------------------------------
TYPE hori11type is array(1 to 13) of std_logic_vector(10 downto 0);
```

```
TYPE hori_11type is array(1 to 14) of  hori11type;

TYPE horitype is array(1 to 13) of integer;
TYPE hori_type is array(1 to 14) of  horitype;

TYPE retype is array(1 to 728) of std_logic_vector(40 downto 0);
TYPE vcoef_decimal_type is array(1 to 728) of integer ;
TYPE red_type is array( 40 downto 0) of integer;
TYPE redec_type is array(1 to 728) of red_type ;


TYPE vcoef_decimal_thr_type is array(1 to 546) of std_logic_vector(40 downto 0);

----------------------------------------------------------------------

type appxthrtype is array (1 to 13) of std_logic_vector(40 downto 0);--14by13
type appx_thrtype is array(1 to 27) of appxthrtype;---actually 14 here--

type vcoef_ups_appx_type1 is array(1 to 13) of std_logic_vector(40 downto 0);
type vcoef_ups_appx_type2 is array(1 to 27) of vcoef_ups_appx_type1 ;

type vcoef_ups_appx_trans_type1 is array(1 to 42) of std_logic_vector(40 downto 0);
type vcoef_ups_appx_trans_type2 is array(1 to 13) of vcoef_ups_appx_trans_type1;

type sc1_appx_type1 is array(1 to 42) of std_logic_vector(56 downto 0);
type sc1_appx_type2 is array(1 to 13) of sc1_appx_type1  ;

type sc1_appx_trans_type1 is array(1 to 40) of std_logic_vector(56 downto 0); ---actually
25--21stapril;
type sc1_appx_trans_type2 is array(1 to 42) of sc1_appx_trans_type1;

type sc1_appx_trans_ups2_type1 is array(1 to 40) of std_logic_vector(56 downto 0);
type sc1_appx_trans_ups2_type2 is array(1 to 42) of sc1_appx_trans_ups2_type1;

type sc2_appx_type1 is array(1 to 40) of std_logic_vector(72 downto 0);
type sc2_appx_type2 is array(1 to 42) of sc2_appx_type1;



begin----------------------------------------------BEGIN OF ARCHITECTURE

process(clk)
-------------------------------------------------DECLARATION OF VARIABLES
variable i,j,k,p,r,m,t,n,b,g,u,f2,v,f11,g1,g3,f12,m1,w1,t1,t2,t3,s1,t4,s2:integer:=1;variable
g5 : integer :=2;
```

```
variable z :integer:=42; variable g2,g4 :integer:=1;
variable w,k1 :integer:=0;
variable d :integer:=1;
variable f3:integer:=1;variable f4:integer:=1;variable f5:integer:=1;
variable f6:integer:=1;variable f7:integer:=1;variable f8:integer:=1;


variable  sx :tdtype;--sx is 13 by 12 matrix
variable  sxint :tdint;--sx is 13 by 12 matrix
variable  sh,shp,shr,shrp:htype;
variable  vxtc:exttc ;---42values+15zeros;
variable  sc :convtype;---sc is 13  by 57 matrix ;
variable  sc_sel :conv_sel;
variable  sc_sel_sub :subsam ;----------------------------------------------here size is 13by6
variable  sc_sel_sub_rext :exttr;-----43by13
variable  sc_sel_sub_rext_trans :sc_sel_subrext_transtype;---13 by 43 matrix but taken as
13 by 58 to add zeros
variable  sc_2,sc_2h :sc_2_typpee ;-------------------------------13 by 58 matrix
variable  sc_2_trans,sc_2h_trans :sc_2_trans_typpee;--------------------58 by 13 matrix
variable  sc_2_trans_sel,sc_2h_trans_sel :sc_2_trans_sel_type;--------------28 by 13
matrix


variable appx,hori :appx_type;

variable  wt : std_logic_vector(24 downto 0):="0000000000000000000000000";
variable wt1,wt2 : std_logic_vector(40 downto
0):="0000000000000000000000000000000000000000000";
variable m2,w2,t1h,t2h,s1h,t3h,s2h,t4h :integer:=1;

---------------------------vd---varia---------------------------------------------------------------------
-------------

variable
ivd,jvd,kvd,pvd,rvd,mvd,tvd,nvd,bvd,gvd,uvd,f2vd,vvd,f11vd,g1vd,g3vd,f12vd,m1vd
:integer:=1;
variable w1vd,t1vd,t2vd,t3vd,s1vd,t4vd,s2vd:integer:=1;
variable g5vd : integer :=2;
variable zvd :integer:=42; variable g2vd,g4vd :integer:=1;
variable wvd,k1vd :integer:=0;
variable dvd :integer:=1;
variable f3vd:integer:=1;variable f4vd:integer:=1;variable f5vd:integer:=1;
variable f6vd:integer:=1;variable f7vd:integer:=1;variable f8vd:integer:=1;

variable  scvd :convtype;---sc is 13  by 57 matrix ;
variable  scvd_sel :conv_sel;
```

variable scvd_sel_sub :subsam ;--here size is 13by6
variable scvd_sel_sub_rext :exttr;-----43by13
variable scvd_sel_sub_rext_trans :sc_sel_subrext_transtype;---13 by 43 matrix but taken as 13 by 58 to add zeros
variable scvd_2,scvd_2h :sc_2_typpee ;---------------------------------13 by 58 matrix
variable scvd_2_trans,scvd_2h_trans :sc_2_trans_typpee;--------------------58 by 13 matrix
variable scvd_2_trans_sel,scvd_2h_trans_sel :sc_2_trans_sel_type;--------------28 by 13 matrix

variable ver,dia :appx_type;

variable wtvd : std_logic_vector(24 downto 0):="0000000000000000000000000";
variable wt1vd,wt2vd : std_logic_vector(40 downto 0):="0000000000000000000000000000000000000000";

variable m2vd,w2vd,t1hvd,t2hvd,s1hvd,t3hvd,s2hvd,t4hvd :integer:=1;

variable p1 :integer:=1;variable p2 :integer:=4;variable p3 :integer:=16;variable p4 :integer:=28;variable p5 :integer:=40;

VARIABLE AHVD_V :AHVDTYPEE;
VARIABLE AHVD_V2 :AHVD_V2TYPE ;

variable ss,ss2,ss3,ss4,ss5,ss6,ss7,ss8,ss9,ss10,ss11,ss12,ss13 :integer :=0;
variable tahvd ,st :integer:=1;
VARIABLE tvdecimal:integer:=455;---assume,29mar

variable zz1,zz2,zz3 :integer :=1;
variable app1:integer:=1;

VARIABLE vcoef :coeftype ;
variable ccc,vv,vv1,vv2 :integer:=1;
variable tccc:integer:=455;
variable vc22,vc23:integer:=0;

----------for decimal conver variables----------
variable vcoefbit :std_logic_vector(1 to 728);

```
variable re       :retype;
variable redec :redec_type;
variable realzero,realone :integer:=0 ;
variable suman1,suman2,vc1s,vc2s :integer:=0;
variable vcoef_1s,vcoef_2s :coeftype;
variable vcoef_decimal :vcoef_decimal_type;
variable bits,bre :integer:=1;
variable bvc2s:integer:=1;
variable brec:integer:=40;
-----------------------using thr-----------------------------------------------
variable thr:std_logic_vector(10 downto 0):="00000100011" ;--------- decimal in 11 bits
---variable thr:integer:=14 ;---------14 decimal in 11 bits

variable vcoef_decimal_thr:vcoef_decimal_thr_type;
variable sthr :integer:=1;
variable sthrd :integer:=273;----assume,actually starts at 1
-----------------------upsamp--------------------------------------------------
variable vcoef_ups_appx,vcoef_ups_hori,vcoef_ups_ver,vcoef_ups_dia :coefupstype;
variable seven :integer:=1;
variable seven2:integer:=1;
variable seven3:integer:=183;
variable seven4:integer:=365;
variable sups:integer:=1;
------------------------------c1u2c3-------------------------------------------
--variable vcoef_ups_appx,vcoef_ups_hori,vcoef_ups_ver,vcoef_ups_dia
:vcoef_ups_appx_type2;
variable hori_11,ver_11,dia_11:hori_11type;
variable appx_thr,hori_thr,ver_thr,dia_thr:appx_thrtype;
variable appx_thr_ups,hori_thr_ups,ver_thr_ups,dia_thr_ups:coefupstype;
variable
vcoef_ups_appx_trans,vcoef_ups_hori_trans,vcoef_ups_ver_trans,vcoef_ups_dia_trans
:vcoef_ups_appx_trans_type2;
variable mres,wres,mres2,wres2 :integer:=1;
variable wtres_appx,wtres_hori,wtres_ver,wtres_dia :std_logic_vector(56 downto
0):="00000000000000000000000000000000000000000000000000000000";
variable wtres2_appx,wtres2_hori,wtres2_ver,wtres2_dia:std_logic_vector(72 downto
0):="000000000000000000000000000000000000000000000000000000000000000000000000
000000";


variable sc1_appx,sc1_hori,sc1_ver,sc1_dia :sc1_appx_type2;
variable sc1_appx_trans,sc1_hori_trans,sc1_ver_trans,sc1_dia_trans
:sc1_appx_trans_type2;
variable
sc1_appx_trans_ups2,sc1_hori_trans_ups2,sc1_ver_trans_ups2,sc1_dia_trans_ups2
:sc1_appx_trans_ups2_type2;
```

```
--variable wtres2_appx,wtres2_hori,wtres2_ver,wtres_dia :std_logic_vector(72 downto
0);


variable sc2_appx,sc2_hori,sc2_ver,sc2_dia,recons :sc2_appx_type2;
variable reconst,saz,saz1,say ,row,reconst1,stem,seez,thori:integer:=1;
variable hori_deci,ver_deci,dia_deci:hori_type;
variable saz2 :integer:=9;
variable sim,sin,sik,set,sent,ser,tre,sit:integer:=1;
variable start:std_logic:='0';
variable  sit1,sitt,sitt1,sf:integer:=1;



--------------------------------------------------------------------------
variable cp1,cpp,cp2,cp3,cp21,cp31:integer:=57;----------cp=cn+30+15;

variable ck:integer:=27;----------ck=(cn+30)-16;

variable ck1:integer:=13;----------ck1=ck/2(if ck is even) or integer of(ck/2)(if ck is
odd);

variable ckdash,ck1dash,cte,ctcf,cta,ct_a,ct_h,ct_h1,ct_v,ct_v1,ct_d:integer:=1;

----------UPSAMPLING----------------------------------------------------------
VARIABLE cp4,cg,cg_g,cg1,cg2,cg3,cg4,cg5,cn1,aa:integer:=1;---INTILIZATION
NOT REQUIRED;

variable ff4,ff5,ff6,ff7,ff8,ff9,ff11,ff12,ff13,ff10 :integer:=1;

variable ff4c,ff5c,ff6c,ff7c,ff8c,ff9c,ff11c,ff12c,ff13c,ff10c :integer:=1;


variable cmby ,cm14:integer:=1;
variable cnbycol ,cncol14,modthcol,cendcol,cstartcol:integer:=1;
variable modth,cstart,cend,godd,geven,g1odd,g1even,prt:integer:=1;--INTILIZATION
NOT REQUIRED
variable pstart,astart,qcntr,bcntr,leftrow,leftcol,cmone,cnone:integer:=1;

-----------------------------------CONSTANTS-------------------------------------------
-------------------------DECLARING DIMENSIONS OF INPUT-----------------------------
------

constant cm:integer:=13;
constant cn:integer:=12;
```

```
begin----------------------------------------------------BEGIN OF PROCESS----------------------



cn1:=cn+1;


---------calculation formulas-----------------------------------------------------------------
cp1:=cn+30+15;----57--befor--conv
cpp:=cp1*cm;
ck:=(cn+30)-16+1;---27--sel--conv
ck1:=ck/2;-----13--samp
cp2:=cm+30;----------------------------------------declare cp21:=cp2+1;
cp21:=cp2+1;--44
cp3:=cm+30+15;--addzer--58
cp31:=cp3+1;-------------59
ckdash:=(cm+30)-16+1;--28--sel-conv
ck1dash:=ckdash/2;---14;


cte:=cm*cn;-------156--total no.of elements
ctcf:=4*(ck1dash*ck1);--728--total no.of coeff.
cta:=ck1dash*ck1;------182

ct_a:=cta+1;------183
ct_h:=ct_a+cta-1;---364
ct_h1:=ct_h+1;----365;

ct_v:=ct_h1+(cta-1);---546
ct_v1:=ct_v+1;-------547

ct_d:=ct_v1+(cta-1);---728

 cp4:=cp2+1;---------------44
-----------UPSAMPLING-------------


 cg:=(ck1dash+ck1dash-1);---27
 cg_g:=cg+1;---28
 cg1:=cg+16-1;----42
 cg2:=cg+1;---27;
```

```
 cg3:=ck1+ck1-1;----25
 cg4:=cg3+1;-----26;
 cg5:=cg3+16-1;----40


---------------------------


cmby:=15/cm;--------0
        cnbycol:=15/cn;
cm14:=cm-14;
        cncol14:=cn-14;

modth:=15 mod cm;----------------------2
         modthcol:=15 mod cn;-------------is 3
cstart:=cm-modth+1;--------12
         cstartcol:=cn-modthcol+1;----------------------12-3+1=10
cend:=modth;--------2
         cendcol:=modthcol;


----------26thapril--------

godd:=(ck1dash +(ck1dash-1)+1)/2;
geven:=(ck1dash +(ck1dash-1))/2;

g1odd:=(ck1+(ck1-1)+1)/2;
g1even:=(ck1+(ck1-1))/2;


pstart:=((cg1-cm)/2)+1;---15
astart:=((cg5-cn)/2)+1;---15

qcntr:=pstart+cm-1;---27
bcntr:=astart+cn-1;---26


leftrow:=((cg1-cm)/2);
leftcol:=((cg5-cn)/2);

cmone:=cm+1;
cnone:=cn+1;


--------------------setting  decompositionFILTTER(LOW) coefficients required----------
sh(1):="0000000001100010";
```

```
sh(2):="1111111011010110";sh(3):="1111110111011010";sh(4):="0000100110001111";
sh(5):="0000010000000001";sh(6):="1101100101110011";sh(7):="1111110010011100";
sh(8):="0111100111001110";
sh(9):=sh(8);sh(10):=sh(7);sh(11):=sh(6);sh(12):=sh(5);sh(13):=sh(4);--here negative
numbers represented in 2's comp.
sh(14):=sh(3);sh(15):="1111111011011010";sh(16):=sh(1);sh(17):="0000000000000000
";
--if(clk'event and clk='1') then
for q in 18 to 57 loop   sh(q):="0000000000000000";end loop;--end if;
--end of required signal declaration


----------------setting 2d matrix(GRAY SCALE IMAGE MATRIX)--------------------------
-----

sxint(1):=(106, 75,  149,  133,  52,  148,  114,  164,  127,  170,  147,  99);
sxint(2):=( 106, 35, 67, 146, 136, 151, 153, 183, 117, 171, 170, 115);
sxint(3):=(106, 54, 44 ,111, 146, 106, 139, 109, 185, 230, 206, 132);
sxint(4):=(120, 125, 130, 70, 119, 145, 167,  85, 196, 250, 208, 115);
sxint(5):= (136, 117, 147,  19, 102, 201, 161, 96, 167, 219, 203, 141);
sxint(6):=(227, 130, 125, 41, 120, 182, 68, 56, 146, 130, 118, 144);
sxint(7):=(255, 185, 140, 138,159, 136, 24, 58, 86, 62, 33,49);
sxint(8):=(99, 141, 94, 151, 122, 65, 21, 74,  53, 123, 100, 1);
sxint(9):=(107, 44,  96,  88,  28, 1, 11, 107, 84, 149, 141,102);
sxint(10):=(157, 68, 65, 54, 60, 83,  41, 32,  3, 58, 47, 7);
sxint(11):=( 99,70, 78, 42, 31, 76, 49, 10, 25, 62, 55, 28);
sxint(12):=( 26, 44, 75, 46, 16, 41,54,  35, 43, 60, 60, 54);
sxint(13):=(55, 21, 16, 49, 67, 77, 75, 30, 42, 31, 38, 51);




if(clk'event and clk='1') then----------------------------------FIRST IF---- FIRST IF-----
FIRST IF----------------------------------------


-------------------------converting in to binary form-----------------------------------------------
for s in 1 to cm loop
  for r in 1 to cn loop
---<slv_sig> = CONV_STD_LOGIC_VECTOR(<int_sig>, <integer_size>);
          sx(s)(r):= CONV_STD_LOGIC_VECTOR(sxint(s)(r),9 );
          end loop;
end loop;


--------------addingcolumns---may14th-----------------------------------------------------------
```

```
   if(cn>=15) then----------------------------------start if;

                          for f3 in 1 to cm loop----------in each row there,assume 13
elements
           ff4c:=0;----------declare this
       for s in 15 downto 1 loop----------assume 15 rows;or above
            ff4c:=ff4c+1;

           vxtc(f3)(ff4c):=sx(f3)(s);


           end loop;

           end loop;


           for f3 in  1 to cm  loop
             ff5c:=ff4c;
             for s in 1 to cn loop
            ff5c:=ff5c+1;----------declre this
           vxtc(f3)(ff5c):=sx(f3)(s);


           end loop;

           end loop;


             for f3 in 1 to cm loop
                   ff6c:=ff5c;
     for s in  cn downto cncol14  loop----------declare cn-14 as variablecncol14

            ff6c:=ff6c+1;----------declare this
           vxtc(f3)(ff6c):=sx(f3)(s);

           end loop;

           end loop;


   elsif(cn<15) then



     for f3  in 1 to cm loop
                    ff7c:=0;
```

```
 for s in  cstartcol to cn loop----------assume start=from noofrows downto modth;  12
to 13 when cm=13
          ----------10 to 12--------correct--15thmay
        ff7c:=ff7c+1;
         vxtc(f3)(ff7c):=sx(f3)(s);

 end loop;
 end loop;


 for s in 1 to cnbycol loop----put one variable for 15/cn---------------------------- 1 to 1


 if(s mod 2=1) then


          ---REVERSE LOOP

           for f3 in 1 to cm loop--------------in each row there,assume 13 elements
                      ff8c:=ff7c;------------------declare this--
     for s in cn downto 1 loop-----------assume 15 rows;or above--13 t0 1
           --------12 downto 1----------corecct------15thmay2008----
          ff8c:=ff8c+1;
                     vxtc(f3)(ff8c):=sx(f3)(s);

     end loop;
          end loop;

       else


          for f3 in 1 to cm loop
               ff8c:=ff7c;------------------declare this--
          for s in  1 to cn  loop---------------FORWARD LOOP

         ff8c:=ff8c+1;----------------declre this
                     vxtc(f3)(ff8c):=sx(f3)(s);

    end loop;
                     end loop;

       end if;
       end loop;----------------------------------------------------------------------
```

```
          for f3  in 1 to cm loop
          ff10c:=ff8c;
         for s in  1 to cn loop----------assume start=from noofrows downto modth;

         -------1 to 12-------------correct------15thmay2008-----
             ff10c:=ff10c+1;
                   vxtc(f3)(ff10c):=sx(f3)(s);

    end loop;
  end loop;
```

```
         for s in  cnbycol downto 1 loop-----------------------------------------------------1 to
1---15thmay2008
  if(s mod 2=1) then


             --REVERSE LOOP

              for f3 in 1 to cm loop---------------in each row there,assume 13 elements
                           ff11c:=ff10c;------------------declare this-
        for s in cn downto 1 loop-----------assume 15 rows;or above
              ------12 downto 1---------15thmay2008-------------------------------
              ff11c:=ff11c+1;
                           vxtc(f3)(ff11c):=sx(f3)(s);

        end loop;
                           end loop;

          else


             for f3 in 1 to cm loop
                           ff11c:=ff10c;------------------declare this-
             for s in  1 to cn  loop----------------FORWARD LOOP

             ff11c:=ff11c+1;----------------declre this
                        vxtc(f3)(ff11c):=sx(f3)(s);
```

```
        end loop;
              end loop;

          end if;
          end loop;----------------------------------------------------------------------------




            for f3  in 1 to cm loop
            ff13c:=ff11c;
          for s in  cendcol downto 1 loop----------assume start=from noofrows downto
modth;-----downto remove now 5.12pm apr2nd

-----------3 downto 1-------------15thmay2008----------------------
              ff13c:=ff13c+1;
                    vxtc(f3)(ff13c):=sx(f3)(s);

   end loop;
  end loop;
end if;------------------------end of if rext;



if( z<=cp1) then -- z starts at 42
  for s in         1 to cm loop
  vxtc(s)(z):="000000000"; --------------------in first clk 1,42 to 13,42 makes zeros
  end loop;
z:=z+1;
end if;



-------------------------------------------------------------------------------UP TO THIS OK


-------------------------------------------------------------------------------------------
---------------CONVOLUTION  now for all col_ext rows-------------------------------------
------
```

```
for s in 1 to cm loop
for n in 1 to cp1 loop    --42+16-1=57;----42+16-1=57;------------cp1:=cn+30+15;----57--
befor--conv
    m:=1; wt:="00000000000000000000000000";
              while (m<=n)loop

                    w:=n+1-m;
              sc(s)(n):=wt+ vxtc(s)(m)*sh(w);----9bits*16bits=25bits
                    wt:=sc(s)(n);
                     m:=m+1;
       end loop;
end loop;
end loop;
```

UP TO THIS OK;

---SELCTINGING CONVOLUTION OUTPUT---

```
for s in 1 to cm loop---------13
  for b in 1 to ck loop------------------------ck:=(cn+30)-16+1;---27--sel--conv
  d:=b+15;
  sc_sel(s)(b):=sc(s)(d);

  end loop;
end loop;
```

UP TO THIS OK;

----------DOWN SAMPLING----------

```
for s in 1 to cm loop----13
   for g1 in 1 to ck1 loop----------------ck1:=ck/2;-----13--samp
          g2:=g1+g1;
          sc_sel_sub(s)(g1):=sc_sel(s)(g2);
          end loop;
end loop;
```

```
-------------------------------------------------------------------------------------------------------------------
---------
-------------------------------------------------------------------------------------------------------------------
-----------
-------------------------------------------------------------------------------------------------------------------
-----------

----------adding  rOws-----------------------------------------------------------

   if(cm>=15) then----------------------------------start if;
       ff4:=0;------------------declare this
      for s in 15 downto 1 loop-----------assume 15 rows;or above
            ff4:=ff4+1;
            for f3 in 1 to ck1 loop--------------in each row there,assume 13 elements
           sc_sel_sub_rext(ff4)(f3):=sc_sel_sub(s)(f3);



           end loop;

           end loop;

           ff5:=ff4;
           for s in  1 to cm  loop
             ff5:=ff5+1;----------------declre this
             for f3 in 1 to ck1 loop
           sc_sel_sub_rext(ff5)(f3):=sc_sel_sub(s)(f3);


           end loop;

           end loop;
     ff6:=ff5;
     for s in  cm downto cm14  loop-----------declare cm-14 as variablecm14
              ff6:=ff6+1;----------------declare this
             for f3 in 1 to ck1 loop
           sc_sel_sub_rext(ff6)(f3):=sc_sel_sub(s)(f3);

           end loop;

           end loop;


   elsif(cm<15) then

     ff7:=0;
```

```
    for s in  cstart to cm loop----------assume start=from noofrows downto modth;  12 to
13 when cm=13
           ff7:=ff7+1;
    for f3  in 1 to ck1 loop
         sc_sel_sub_rext(ff7)(f3):=sc_sel_sub(s)(f3);


  end loop;
  end loop;

  ff8:=ff7;------------------declare this--
 for s in 1 to cmby loop----put one variable for 15/cm---------------------------- 1 to 1


 if(s mod 2=1) then


          ---REVERSE LOOP
      for s in cm downto 1 loop-----------assume 15 rows;or above--13 t0 1
           ff8:=ff8+1;
           for f3 in 1 to ck1 loop--------------in each row there,assume 13 elements
           sc_sel_sub_rext(ff8)(f3):=sc_sel_sub(s)(f3);


      end loop;
           end loop;

        else


           for s in  1 to cm  loop---------------FORWARD LOOP
           ff8:=ff8+1;----------------declre this
           for f3 in 1 to ck1 loop
           sc_sel_sub_rext(ff8)(f3):=sc_sel_sub(s)(f3);

      end loop;
                      end loop;

        end if;
        end loop;-------------------------------------------------------------------------

        ff10:=ff8;
        for s in  1 to cm loop----------assume start=from noofrows downto modth;

        ff10:=ff10+1;
          for f3  in 1 to ck1 loop
```

```
            sc_sel_sub_rext(ff10)(f3):=sc_sel_sub(s)(f3);


   end loop;
end loop;


         ff11:=ff10;------------------declare this-
        for s in  cmby downto 1 loop-----------------------------------------------------
 if(s mod 2=1) then


            --REVERSE LOOP
       for s in cm downto 1 loop-----------assume 15 rows;or above
             ff11:=ff11+1;
             for f3 in 1 to ck1 loop--------------in each row there,assume 13 elements
             sc_sel_sub_rext(ff11)(f3):=sc_sel_sub(s)(f3);


       end loop;
                        end loop;

         else


            for s in  1 to cm  loop---------------FORWARD LOOP
            ff11:=ff11+1;----------------declre this
            for f3 in 1 to ck1 loop
            sc_sel_sub_rext(ff11)(f3):=sc_sel_sub(s)(f3);


      end loop;
            end loop;

        end if;
        end loop;-------------------------------------------------------------------------------



        ff13:=ff11;
        for s in  cend downto 1 loop----------assume start=from noofrows downto modth;-
----downto remove now 5.12pm apr2nd

        ff13:=ff13+1;
          for f3  in 1 to ck1 loop
            sc_sel_sub_rext(ff13)(f3):=sc_sel_sub(s)(f3);
```

```
    end loop;
   end loop;
end if;------------------------end of if rext;

-------------------------------------------------------------------------------------------------------------
-----------
-------------------------------------------------------------------------------------------------------------
------------

-------


-----------------------------finding trans------pose of y2  i.e above -------------------------------
-----

for c1 in 1 to cm  loop         -------------c1 also used in for loop only---trans--1(line1092,)
   for s in 1 to cp2 loop-----------------------cp2:=cm+30;--ext?43--
   sc_sel_sub_rext_trans(c1)(s):=sc_sel_sub_rext(s)(c1);------13 by 43

   end loop;
end loop;
--------adding zeros to sc_sel_sub_rext_trans and to sh ------------------



for  s in 1 to cm loop-----------13
    for r1 in cp21 to cp3 loop--------------------------cp3:=cm+30+15;--addzer--58---------
cp21:=cp2+1;--44
            sc_sel_sub_rext_trans(s)(r1):="000000000000000000000000";

            end loop;
end loop;
for q in cp3 to cp31 loop   sh(q):="0000000000000000";
shp(q):="0000000000000000";end loop;-----------
```



```
-----------------------------------------------------------------------------------------------------------

-----now do col-conv with wol-------------------------------------------------------------------------------
---------------

for s in 1 to cm loop--------------------13
for n in 1 to cp3 loop    --43+16-1=58;-------------cp3:=cm+30+15;--addzer?58-------------
-----------------
    m1:=1; wt1:="00000000000000000000000000000000000000000000";---declare
m1,wt1,w1,t1,sc_2 matrix,sc_2_out matrix

                    while (m1<=n)loop

                     w1:=n+1-m1;
              sc_2(s)(n):=wt1+ sc_sel_sub_rext_trans(s)(m1)*sh(w1);----
sc_sel_sub_rext_trans is 13 by 43 i.e 13 by 58 after adding zeros
                     wt1:=sc_2(s)(n);
                      m1:=m1+1;
       end loop;
end loop;
end loop;




-----------------finding trans of sc_2 -------------------------

-----------------finding trans of sc_2 -------------------------
for c1 in 1 to cp3  loop        -------------c1  used in for loop only---------cp3:=cm+30+15;--
addzer?58-----------------------------
  for s in 1 to cm loop        ----------------sc_2 is 13 by 58------------13
  sc_2_trans(c1)(s):=sc_2(s)(c1);--------------sc_2_trans is 58 by 13 matrix

  end loop;
end loop;

------------------selctive(selecting rows) conv of sc_2_trans-------------------
for s in 1 to ckdash loop-----------------------------------------------------------
ckdash:=(cm+30)-16+1;--28--sel-conv
    s1:=s+15;                 -----------declare s1
    for r1 in 1 to cm loop-------------13

           sc_2_trans_sel(s)(r1):=sc_2_trans(s1)(r1);-----sc_2_trans_sel is 28 by 13 matrix
```

```
            end loop;
end loop;

      ------to see the sc_2_trans_sel------



------------------now do dow-----------------------------------

for s in 1 to ck1dash loop--------------------ck1dash:=ckdash/2;---14;


   s2:=s+s;
         for r1 in 1 to ck1 loop----------------------13
   appx(s)(r1):=sc_2_trans_sel(s2)(r1);-------------declare s2,appx()(),appx_out

   end loop;
end loop;









----------------------now hor------------------------------


for s in 1 to 6 loop shp(s):="0000000000000000";end loop;
shp(7):="1110100101011111";shp(8):="0100001111100000";shp(9):="10111100001000
00";shp(10):="0001011010100001";
for s in 11 to 16 loop shp(s):="0000000000000000";end loop;
--adding zeros upto 58  (15+13+15)+16-1=58;
for s in 17 to 58 loop  shp(s):="0000000000000000";end loop;




-----now do col-conv with gh-------------------------------------------------------------------
-------------

for s in 1 to cm loop----------------------13
```

```
for n in 1 to cp3 loop    --42+16-1=57;----------------cp3:=cm+30+15;--addzer?58----------
-
    m2:=1; wt2:="00000000000000000000000000000000000000000000";---declare
m2,wt2,w2,t1h,sc_2h matrix,sc_2h_out matrix

                    while (m2<=n)loop

                    w2:=n+1-m2;
              sc_2h(s)(n):=wt2+ sc_sel_sub_rext_trans(s)(m2)*shp(w2);----
sc_sel_sub_rext_trans is 13 by 43 i.e 13 by 58 after adding zeros
                    wt2:=sc_2h(s)(n);



                     m2:=m2+1;                                    ----------write shp values
      end loop;-------------27THMAR2008----IMP-IMP-IMP-IMP
end loop;
end loop;




-----------------finding trans of sc_2h -------------------------
for c1 in 1 to cp3  loop       -------------c1  used in for loop only----------cp3:=cm+30+15;-
-addzer?58-----------
  for s in 1 to cm loop       ----------------sc_2h is 13 by 58---------13
  sc_2h_trans(c1)(s):=sc_2h(s)(c1);--------------sc_2h_trans is 58 by 13 matrix

  end loop;
end loop;


        ---to see the sc_2_trans-----

-----------------selctive(selecting rows) conv of sc_2h_trans-------------------
for s in 1 to ckdash loop--------------ckdash:=(cm+30)-16+1;--28--sel-conv
    s1h:=s+15;                ----------declare s1h
    for r1 in 1 to cm loop

           sc_2h_trans_sel(s)(r1):=sc_2h_trans(s1h)(r1);-----sc_2_trans_sel is 28 by 13
matrix

           end loop;
end loop;
```

------to see the sc_2_trans_sel------

----------now do dow----------

```
for s in 1 to ck1dash loop----------ck1dash:=ckdash/2;---14;
   s2h:=s+s;
         for r1 in 1 to ck1 loop
   hori(s)(r1):=sc_2h_trans_sel(s2h)(r1);----------declare s2h,hori()(),hori_out

   end loop;
end loop;



--convolution  now for all col_ext rows



for s in 1 to cm loop
for n in 1 to cp1 loop    --42+16-1=57;----------cp1:=cn+30+15;----57--befor--conv
    m:=1; wt:="000000000000000000000000";wtvd:=wt;
              while (m<=n)loop

                    w:=n+1-m;

                    scvd(s)(n):=wtvd+ vxtc(s)(m)*shp(w);

                    wtvd:=scvd(s)(n);
                     m:=m+1;
      end loop;
end loop;
end loop;
```

----------UP TO THIS OK;

---selective volu (----------

for s in 1 to cm loop----------13

```
  for b in 1 to ck loop-----------------------ck:=(cn+30)-16+1;---27--sel--conv
  d:=b+15;
 scvd_sel(s)(b):=scvd(s)(d);
  end loop;
end loop;



-------------------------------------------------------------------------------------------------------
UP TO THIS OK;
--now dow samp-----------------------------------------------------------------------------
------1st last values i have to check i.e v starts at what value ....assume v starts at 2 i.e 2 to
27
---------------------------------------------



for s in 1 to cm loop
   for g1 in 1 to ck1 loop------------ck1:=ck/2;-----13--samp
           g2:=g1+g1;

           scvd_sel_sub(s)(g1):=scvd_sel(s)(g2);----down-2(line1289)
           end loop;
end loop;



--end if;
--sc_sel_sub is 13 by 13 matrix ;




-------------------------------------------------------------------------------------------------------
----------
-------------------------------------------------------------------------------------------------------
------------
-------------------------------------------------------------------------------------------------------
------------


---------------addingrws-----------------------------------------------------------------

    if(cm>=15) then----------------------------------start if;
        ff4:=0;------------------declare this
       for s in 15 downto 1 loop----------assume 15 rows;or above
             ff4:=ff4+1;
```

```
          for f3 in 1 to ck1 loop----------in each row there,assume 13 elements

         scvd_sel_sub_rext(ff4)(f3):=scvd_sel_sub(s)(f3);


         end loop;

         end loop;

         ff5:=ff4;
         for s in  1 to cm  loop
           ff5:=ff5+1;----------------declre this
          for f3 in 1 to ck1 loop

         scvd_sel_sub_rext(ff5)(f3):=scvd_sel_sub(s)(f3);


         end loop;

         end loop;
    ff6:=ff5;
    for s in  cm downto cm14  loop-----------declare cm-14 as variablecm14
           ff6:=ff6+1;----------------declare this
          for f3 in 1 to ck1 loop

         scvd_sel_sub_rext(ff6)(f3):=scvd_sel_sub(s)(f3);

         end loop;

         end loop;


  elsif(cm<15) then

    ff7:=0;
    for s in  cstart to cm loop----------assume start=from noofrows downto modth;  12 to
13 when cm=13
          ff7:=ff7+1;
    for f3  in 1 to ck1 loop

         scvd_sel_sub_rext(ff7)(f3):=scvd_sel_sub(s)(f3);

  end loop;
  end loop;

  ff8:=ff7;----------declare this--
```

```
for s in 1 to cmby loop----put one variable for 15/cm---------------------------- 1 to 1


if(s mod 2=1) then


          ---REVERSE LOOP
      for s in cm downto 1 loop-----------assume 15 rows;or above--13 t0 1
            ff8:=ff8+1;
            for f3 in 1 to ck1 loop--------------in each row there,assume 13 elements

                        scvd_sel_sub_rext(ff8)(f3):=scvd_sel_sub(s)(f3);

      end loop;
            end loop;

        else


           for s in  1 to cm  loop---------------FORWARD LOOP
           ff8:=ff8+1;----------------declre this
           for f3 in 1 to ck1 loop

                        scvd_sel_sub_rext(ff8)(f3):=scvd_sel_sub(s)(f3);

     end loop;
                        end loop;

       end if;
       end loop;-----------------------------------------------------------------------

       ff10:=ff8;
       for s in  1 to cm loop----------assume start=from noofrows downto modth;

       ff10:=ff10+1;
         for f3  in 1 to ck1 loop

                scvd_sel_sub_rext(ff10)(f3):=scvd_sel_sub(s)(f3);

  end loop;
end loop;


        ff11:=ff10;------------------declare this-
       for s in  cmby downto 1 loop-----------------------------------------------------
if(s mod 2=1) then
```

```
            --REVERSE LOOP
        for s in cm downto 1 loop----------assume 15 rows;or above
              ff11:=ff11+1;
              for f3 in 1 to ck1 loop--------------in each row there,assume 13 elements

                              scvd_sel_sub_rext(ff11)(f3):=scvd_sel_sub(s)(f3);

        end loop;
                              end loop;

          else


              for s in  1 to cm  loop---------------FORWARD LOOP
              ff11:=ff11+1;----------------declre this
              for f3 in 1 to ck1 loop

                             scvd_sel_sub_rext(ff11)(f3):=scvd_sel_sub(s)(f3);

       end loop;
              end loop;

          end if;
          end loop;--------------------------------------------------------------------------------



          ff13:=ff11;
          for s in  cend downto 1 loop----------assume start=from noofrows downto modth;-
----downto remove now 5.12pm apr2nd

          ff13:=ff13+1;
            for f3  in 1 to ck1 loop

                    scvd_sel_sub_rext(ff13)(f3):=scvd_sel_sub(s)(f3);

    end loop;
   end loop;
end if;------------------------end of if rext;



for c1 in 1 to cm  loop        -------------c1 also used in for loop only---trans--1(line1092,)
  for s in 1 to cp2 loop-----------------------cp2:=cm+30;--ext?43--
```

```
        scvd_sel_sub_rext_trans(c1)(s):=scvd_sel_sub_rext(s)(c1);
  end loop;
end loop;



for  s in 1 to cm loop----------13
   for r1 in cp21 to cp3 loop--------------------------cp3:=cm+30+15;--addzer--58---------
cp21:=cp2+1;--44

        scvd_sel_sub_rext_trans(s)(r1):="000000000000000000000000";
        end loop;
end loop;
for q in cp3 to cp31 loop   sh(q):="0000000000000000";
shp(q):="0000000000000000";end loop;----------cp31:=cp3+1;



-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------



-----now do col-conv with low-----------------------------------------------------------------
--------------


for s in 1 to cm loop-------------------13
for n in 1 to cp3 loop    --43+16-1=58;-------------cp3:=cm+30+15;--addzer?58-------------
-----------------
    m1:=1; wt1:="000000000000000000000000000000000000000000000000";---declare
m1,wt1,w1,t1,sc_2 matrix,sc_2_out matrix
            wt1vd:=wt1;
                  while (m1<=n)loop

                   w1:=n+1-m1;


      scvd_2(s)(n):=wt1vd+ scvd_sel_sub_rext_trans(s)(m1)*sh(w1);----
sc_sel_sub_rext_trans is 13 by 43 i.e 13 by 58 after adding zeros
                   wt1vd:=scvd_2(s)(n);

                   m1:=m1+1;
      end loop;
```

```
end loop;
end loop;



------------------finding trans of sc_2 -------------------------
for c1 in 1 to cp3  loop        ------------c1  used in for loop only---------cp3:=cm+30+15;--
addzer?58-----------------------------
  for s in 1 to cm loop        ----------------sc_2 is 13 by 58------------13

        scvd_2_trans(c1)(s):=scvd_2(s)(c1);--------------sc_2_trans is 58 by 13 matrix
  end loop;
end loop;




------------------selctive(selecting rows) conv of sc_2_trans-------------------
for s in 1 to ckdash loop-----------------------------------------------------------
ckdash:=(cm+30)-16+1;--28--sel-conv
    s1:=s+15;                ------------declare s1
    for r1 in 1 to cm loop-------------13



          scvd_2_trans_sel(s)(r1):=scvd_2_trans(s1)(r1);-----sc_2_trans_sel is 28 by 13
matrix
          end loop;
end loop;




-------------------now do dow-----------------------------------

for s in 1 to ck1dash loop--------------------ck1dash:=ckdash/2;---14;


   s2:=s+s;
        for r1 in 1 to ck1 loop----------------------13

        ver(s)(r1):=scvd_2_trans_sel(s2)(r1);-------------declare s2vd,appx()(),v_out
   end loop;
end loop;
```

```
-----now do col-conv with gh-------------------------------------------------------------------------------

for s in 1 to cm loop----------------------13
for n in 1 to cp3 loop    --42+16-1=57;----------------cp3:=cm+30+15;--addzer?58----------
-
    m2:=1; wt2:="0000000000000000000000000000000000000000000000";---declare
m2,wt2,w2,t1h,sc_2h matrix,sc_2h_out matrix
              wt2vd:=wt2;
                    while (m2<=n)loop

                    w2:=n+1-m2;

                    scvd_2h(s)(n):=wt2vd+ scvd_sel_sub_rext_trans(s)(m2)*shp(w2);----
sc_sel_sub_rext_trans is 13 by 43 i.e 13 by 58 after adding zeros
                    wt2vd:=scvd_2h(s)(n);



                     m2:=m2+1;
       end loop;
end loop;
end loop;

-----------------finding transpose of sc_2h -------------------------
for c1 in 1 to cp3  loop        -------------c1  used in for loop only----------cp3:=cm+30+15;-
-addzer?58-----------
  for s in 1 to cm loop        ----------------sc_2h is 13 by 58---------13

          scvd_2h_trans(c1)(s):=scvd_2h(s)(c1);--------------sc_2h_trans is 58 by 13 matrix
  end loop;
end loop;


-----------------selctive(selecting rows) conv of sc_2h_trans-------------------
for s in 1 to ckdash loop--------------ckdash:=(cm+30)-16+1;--28--sel-conv
    s1h:=s+15;                -----------declare s1h
    for r1 in 1 to cm loop


           scvd_2h_trans_sel(s)(r1):=scvd_2h_trans(s1h)(r1);-----sc_2_trans_sel is 28 by
13 matrix
           end loop;
end loop;
```

```
------------------now do DOWNNSAMPLING-----------------------------------

for s in 1 to ck1dash loop-------------------------ck1dash:=ckdash/2;---14;
   s2h:=s+s;
         for r1 in 1 to ck1 loop

         dia(s)(r1):=scvd_2h_trans_sel(s2h)(r1);-------------declare s2h,hori()(),hori_out
   end loop;
end loop;


for s in 1 to ck1dash loop------row wise--14
  for s1 in 1 to ck1 loop---13

hori_11(s)(s1)(10 downto 0):=hori(s)(s1)(40 downto 30);-----declre
hori_11()(),ver_11,dia_11
ver_11(s)(s1)(10 downto 0):=ver(s)(s1)(40 downto 30);-----declre
hori_11()(),ver_11,dia_11
dia_11(s)(s1)(10 downto 0):=dia(s)(s1)(40 downto 30);-----declre
hori_11()(),ver_11,dia_11

  end loop;
end loop;



for s in 1 to ck1dash loop------row wise--14
 for s1 in 1 to ck1 loop--13


 if(hori_11(s)(s1)<"00000000000")then
hori_11(s)(s1):=not( hori_11(s)(s1));end if;---into 1s coplement;
hori_11(s)(s1):=hori_11(s)(s1) + "00000000001";-----adding 1 i.e into 2's complement
i.e to take magnitude

if(ver_11(s)(s1)<"00000000000")then
ver_11(s)(s1):=not( ver_11(s)(s1));end if;
ver_11(s)(s1):=ver_11(s)(s1) + "00000000001";-----adding 1 i.e into 2's complement i.e
to take magnitude
 if(dia_11(s)(s1)<"00000000000")then
dia_11(s)(s1):=not( dia_11(s)(s1));end if;
dia_11(s)(s1):=dia_11(s)(s1) + "00000000001";-----adding 1 i.e into 2's complement i.e
to take magnitude
```

```
end loop;

end loop;----d

for s in 1 to ck1dash loop------row wise
  for s1 in 1 to ck1 loop

if(hori_11(s)(s1)>=thr)then hori_thr(s)(s1):=hori(s)(s1);else
hori_thr(s)(s1):="00000000000000000000000000000000000000000000";end if;
if(ver_11(s)(s1)>=thr)then ver_thr(s)(s1):=ver(s)(s1);else
ver_thr(s)(s1):="00000000000000000000000000000000000000000000";end if;
if(dia_11(s)(s1)>=thr)then dia_thr(s)(s1):=dia(s)(s1);else
dia_thr(s)(s1):="00000000000000000000000000000000000000000000";end if;


end loop;

end loop;----declare hori_thr()();--as 14by13--40 downto 0 matrix;

--if(tre<=13) then
--hori_out<=hori(1)(tre);

--tre:=tre+1;
--end if;


----***if(thori<=13)then---*****************************
--appx_thr_out<=appx(prt)(thori);
--hori_thr_out<=hori_thr(prt)(thori);
--ver_thr_out<=ver_thr(prt)(thori);
---dia_thr_out<=dia_thr(prt)(thori);



---thori:=thori+1;
---if(thori=14) then prt:=prt+1;thori:=1 ;end if;--declare prt,strats at 1;
---******end if;---************************************

---------------------------------------------------------------------------------------------
```

```
-------------------------now up samp 19 mar--19thapril-----------------------------------------------
------
-----------------------------------------------------------------------------------------
-----appx ups(row)----------------------------results in (14+13)by13--------------------------
----------------------------------
sit:=1;
for s in 1 to godd loop------------14rows+13 gaps ,in that  these are 14 odd indexed
numbers---14---godd:=[ck1dash +(ck1dash-1)+1]/2

            for sapp1 in 1 to ck1 loop--13


                    appx_thr_ups(sit)(sapp1):=appx(s)(sapp1);----DECALRE
APPX_THR_UPS AS 27BY13 VARIABLE;
                    hori_thr_ups(sit)(sapp1):=hori_thr(s)(sapp1);----DECALRE
APPX_THR_UPS AS 27BY13 VARIABLE;
                    ver_thr_ups(sit)(sapp1):=ver_thr(s)(sapp1);----DECALRE
APPX_THR_UPS AS 27BY13 VARIABLE;
                    dia_thr_ups(sit)(sapp1):=dia_thr(s)(sapp1);----DECALRE
APPX_THR_UPS AS 27BY13 VARIABLE;



     end loop;
                  sit:=sit+2;-----sits starts at 1;



          end loop;

          for s in 1 to geven loop                    -----these are 13 even numbers  ----------23rd
april;-----13
          --elsif(s mod 2=0) then
          sit1:=2*s;----------sit1 starts at 1 ,ends at 26;
             for sapp2 in 1 to ck1 loop


appx_thr_ups(sit1)(sapp2):="00000000000000000000000000000000000000000000";

hori_thr_ups(sit1)(sapp2):="00000000000000000000000000000000000000000000";
            ver_thr_ups(sit1)(sapp2):="00000000000000000000000000000000000000000000";
            dia_thr_ups(sit1)(sapp2):="00000000000000000000000000000000000000000000";
             end loop;


end loop;

---setting reconstruction filter(LOW) coefficients required---------------------------------
```

```
shr(1):="0000000000000000";------------------declre shr
shr(2):=shr(1);shr(3):=shr(1);shr(4):=shr(1);
shr(5):=shr(1);shr(6):=shr(1);shr(7):="0001011010100001";shr(8):="0100001111100000
";
shr(9):=shr(8);shr(10):=shr(7);shr(11):=shr(1);shr(12):=shr(1);shr(13):=shr(1);--here
negative numbers represented in 2's comp.
shr(14):=shr(1);shr(15):=shr(1);shr(16):=shr(1);shr(17):="0000000000000000";
---adding zeros--------------------------------------------------------------------------

for s in 17 to cg1 loop
shr(s):="0000000000000000";---lowp-recons
end loop;
---setting reconstruction filter(HIGH) required-----------------------------------------

shrp(1):="0000000001100010";
shrp(2):="0000000100101010";shrp(3):="1111110111011010";shrp(4):="111101100111
0001";
shrp(5):="0000010000000001";shrp(6):="0010011010001101";shrp(7):="111111001001
1100";shrp(8):="1000011000110010";
shrp(9):="0111100111001110";shrp(10):="0000001101100100";shrp(11):="1101100101
110011";shrp(12):="1111101111111111";
shrp(13):="0000100110001111";--here negative numbers represented in 2's comp.
shrp(14):="0000001000100110";shrp(15):="1111111011010110";shrp(16):="111111111
0011110";shrp(17):="0000000000000000";
---adding zeros--------------------------------------------------------------------
for s in 17 to cg1 loop
shrp(s):="0000000000000000";---hiGhp-recons
end loop;
---transpos appx first-------------------u----c1u2c3-for evry on a,h,v,d;
-----------------finding trans of vcoef_ups_appx,hori,ver,dia and adding 15 zeros to each
row-------------------------
for c1 in 1 to ck1  loop        -------------c1  used in for loop only
  for s in 1 to cg loop        ----------------sc_2 is 27 by 13--------
  vcoef_ups_appx_trans(c1)(s):=appx_thr_ups(s)(c1);---vcoef_ups_trans is 13 by 27
matrixVERY VERY IMP but declare as 13by42
        vcoef_ups_hori_trans(c1)(s):=hori_thr_ups(s)(c1);-----declare
vcoef_ups_appx,hori,dia,ver as 13by 27
        vcoef_ups_ver_trans(c1)(s):=ver_thr_ups(s)(c1);
        vcoef_ups_dia_trans(c1)(s):=dia_thr_ups(s)(c1);
  end loop;
end loop;
----------------------UPSAMPLINGPROCESS-----------------------------------------------------
--------------
--adding zeros-----------------------------------------------------------------------------------
---
for sz in 1 to ck1 loop
```

```
  for s in cg to cg1 loop--28 to 42
        vcoef_ups_appx_trans(sz)(s):="00000000000000000000000000000000000000000
0";
        vcoef_ups_hori_trans(sz)(s):="00000000000000000000000000000000000000000
";
        vcoef_ups_dia_trans(sz)(s):="0000000000000000000000000000000000000000"
;
        vcoef_ups_ver_trans(sz)(s):="00000000000000000000000000000000000000000
";
        end loop;
end loop;

----------------------------c1(col convol) of all a,h,v,d-----------------------------------

for s in 1 to ck1 loop-------------------------------declare required variables here--13
for n in 1 to cg1 loop    --27+16-1=42;--42
    mres:=1;
wtres_appx:="000000000000000000000000000000000000000000000000000000000000";w
tres_hori:=wtres_appx;wtres_ver:=wtres_appx;wtres_dia:=wtres_appx;
             while (mres<=n)loop

                    wres:=n+1-mres;
      ---appx conv1-----
             sc1_appx(s)(n):=wtres_appx+ vcoef_ups_appx_trans(s)(mres)*shr(wres);----
41bits*16bits=57bits---low
                    wtres_appx:=sc1_appx(s)(n);----results in 13by42
                   ---hori conv1----
                    sc1_hori(s)(n):=wtres_hori+ vcoef_ups_hori_trans(s)(mres)*shrp(wres);-
---41bits*16bits=57bits---hig
                    wtres_hori:=sc1_hori(s)(n);-----13by42
                    ---ver conv1--
                    sc1_ver(s)(n):=wtres_ver+ vcoef_ups_ver_trans(s)(mres)*shr(wres);----
41bits*16bits=57bits-------low
                    wtres_ver:=sc1_ver(s)(n);------13by42
                    ---dia conv1--
                    sc1_dia(s)(n):=wtres_dia+ vcoef_ups_dia_trans(s)(mres)*shrp(wres);----
41bits*16bits=57bits-------hig
                    wtres_dia:=sc1_dia(s)(n);------13by42

                    mres:=mres+1;
       end loop;
end loop;
end loop;


----------------------------trans of all of above a,h,v,d-----results in 42by13----------------
```

```
for c1 in 1 to cg1  loop        ----------c1  used in for loop only
  for s in 1 to ck1 loop        ----------sc_2 is 27 by 13
  sc1_appx_trans(c1)(s):=sc1_appx(s)(c1);----------vcoef_ups_trans is 42 by 13 matrix
          sc1_hori_trans(c1)(s):=sc1_hori(s)(c1);----------declare
vcoef_ups_appx,hori,dia,ver as 13by 27
          sc1_ver_trans(c1)(s):=sc1_ver(s)(c1);
          sc1_dia_trans(c1)(s):=sc1_dia(s)(c1);
  end loop;
end loop;
----------column ups of all above a,h,v,d------42by25----------
--------


----------column ups of all above a,h,v,d------42by25----------
--------
sitt:=1;
for s in 1 to g1odd loop-------13 odd indexed numbers---
  --13 cols,12 gaps so total 25, in it 13 odd,12 even indexed numbers;

          for sd in 1 to cg1 loop----------sc1_appx_trans is a 42by13 i.e only 13
columns
  sc1_appx_trans_ups2(sd)(sitt):=sc1_appx_trans(sd)(s);----declare sc1_appx_trans_ups2
as 42by25
  sc1_hori_trans_ups2(sd)(sitt):=sc1_hori_trans(sd)(s);
  sc1_ver_trans_ups2(sd)(sitt):=sc1_ver_trans(sd)(s);
  sc1_dia_trans_ups2(sd)(sitt):=sc1_dia_trans(sd)(s);
          end loop;

          sitt:=sitt+2;----starts at 1,ends at 25

end loop;


for s in 1 to g1even loop-----12 even indexed numbers
    sitt1:=s*2;----declare sitt1,strats at 1,ends at 24;
    for sd2 in 1 to cg1 loop

sc1_appx_trans_ups2(sd2)(sitt1):="000000000000000000000000000000000000000000000
00000000000000";---42by25----but declare as 42by40

sc1_hori_trans_ups2(sd2)(sitt1):="000000000000000000000000000000000000000000000
0000000000000";---42by25

sc1_ver_trans_ups2(sd2)(sitt1):="000000000000000000000000000000000000000000000
0000000000000";
```

```
sc1_dia_trans_ups2(sd2)(sitt1):="0000000000000000000000000000000000000000000000000000000000000";--57bits
  end loop;
end loop;
-------------------------now adding zeros-----to sc1_appx_trans_ups2(42by25)-----------------------
-------------------------------25+16-1=40-----so add zeros-----42by(25+15zeros)-----------------------------
for s in 1 to cg1 loop
  for sz in cg4 to cg5 loop----adding 15 zeros---- 26 to 40
        sc1_appx_trans_ups2(s)(sz):="0000000000000000000000000000000000000000000000000000000000000";---57bits----42by(25+15zeros)

sc1_hori_trans_ups2(s)(sz):="0000000000000000000000000000000000000000000000000000000000000";---57bits
        sc1_ver_trans_ups2(s)(sz):="0000000000000000000000000000000000000000000000000000000000000";---57bits

sc1_dia_trans_ups2(s)(sz):="0000000000000000000000000000000000000000000000000000000000000";---57bits
  end loop;
end loop;
-------------------------now conv2 of all above,a,h,v,d-------------------------
for s in 1 to cg1 loop-----------------------------declare required variables here--42
for n in 1 to cg5 loop    --25+16-1=40;--40
    mres2:=1;
wtres2_appx:="0000000000000000000000000000000000000000000000000000000000000";wtres2_hori:=wtres2_appx;wtres2_ver:=wtres2_appx;wtres2_dia:=wtres2_appx;
            while (mres2<=n)loop

                  wres2:=n+1-mres2;  ----------------declare required varaibles
     ---appx conv1-----
            sc2_appx(s)(n):=wtres2_appx+ sc1_appx_trans_ups2(s)(mres2)*shr(wres2);----57bits*16bits=73bits---low--low
                  wtres2_appx:=sc2_appx(s)(n);----results in 42by40
                 ---hori conv1----
                  sc2_hori(s)(n):=wtres2_hori+ sc1_hori_trans_ups2(s)(mres2)*shr(wres2);----57bits*16bits=73bits---hig--low
                  wtres2_hori:=sc2_hori(s)(n);-----42by40
                  ---ver conv1--
                  sc2_ver(s)(n):=wtres2_ver+ sc1_ver_trans_ups2(s)(mres2)*shrp(wres2);----57bits*16bits=73bits-------low--hig
                  wtres2_ver:=sc2_ver(s)(n);------42by40
                  ---dia conv1--
```

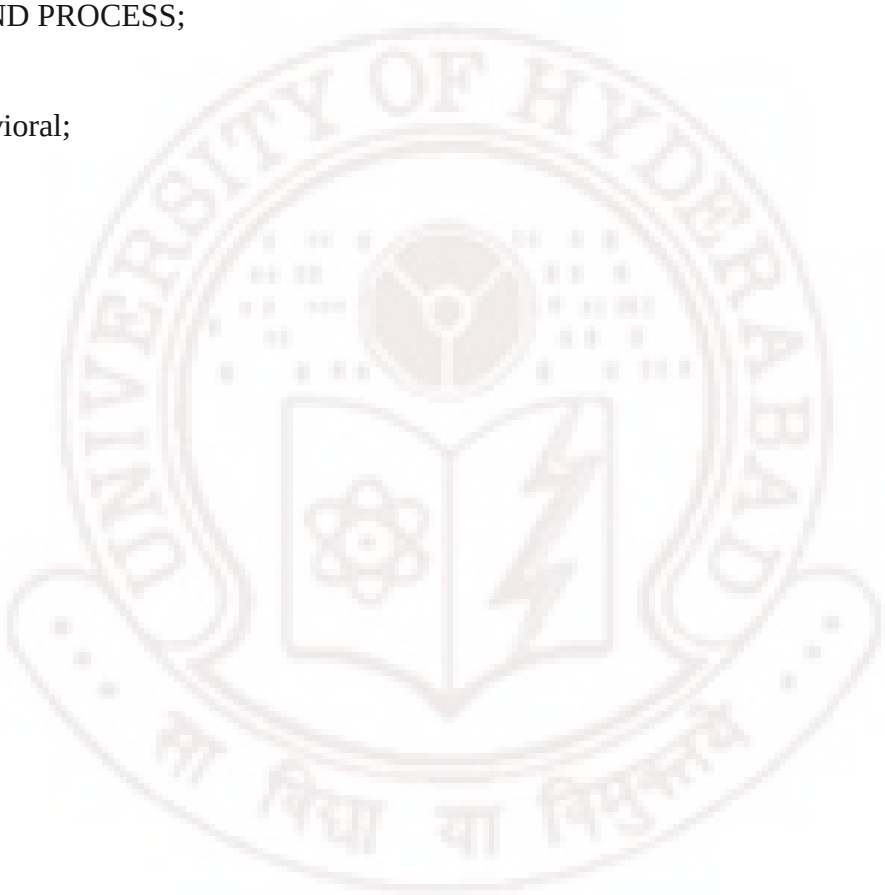
```
                    sc2_dia(s)(n):=wtres2_dia+ sc1_dia_trans_ups2(s)(mres2)*shrp(wres2);-
---57bits*16bits=73bits-------hig--hig
                    wtres2_dia:=sc2_dia(s)(n);------42by40

                    mres2:=mres2+1;
      end loop;
end loop;
end loop;

pstart:=((cg1-cm)/2)+1;---15
astart:=((cg5-cn)/2)+1;---15

qcntr:=pstart+cm-1;---27
bcntr:=astart+cn-1;---26


--------------------------------addition of all ,above a,h,v,d-------------------------
for s in pstart to qcntr loop
stem:=s-leftrow;--------stem,starts at 1;--14 is leftrow
  --if(stem<=13) then-----------stem strts at 1;
     for sz in astart to bcntr loop
       --if(seez<=12)then----------seez strats at 1;
                        seez:=sz-leftcol;---------declare seez ,starts at 1;----14 is leftcol

recons(stem)(seez):=sc2_appx(s)(sz)+sc2_hori(s)(sz)+sc2_ver(s)(sz)+sc2_dia(s)(sz);---
declare recons as 42by40 of 73bits;



            -- end if;
     end loop;

  --end if;
end loop;
if(row<=cmone and reconst1<=cnone)then------------13by12=156--14,13
 if(reconst1=cnone) then reconst1:=1;row:=row+1;end if;---declare row ,row starts at 1;--
----12elements
 re_out_row<=recons(row)(reconst1);---declare re_out_row same as re_out;
 reconst1:=reconst1+1;--------------declere reconst1 ,starts at 1;
 end if;

end if;--------------------END OF FIRST IF----------END OF FIRST IF----------------------
-------------------------------
end process;
end Behavioral;
```

TEST BENCH :

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:12:02 02/29/2008
-- Design Name:
-- Module Name:    testbench_ahvd - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
use ieee.std_logic_TEXTIO.all;
use ieee.std_logic_arith.all;

use std.textio.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity compr_28th_mar is
end compr_28th_mar;

architecture Behavioral of compr_28th_mar is

component compr is
port( clk :in std_logic;re_out_row:out std_logic_vector(72 downto 0)




);
end component;
```

```
--Inputs
	SIGNAL clk :  std_logic ;

--Outputs
	--SIGNAL re_out_row :  std_logic_vector(72 downto 0);
	SIGNAL re_out_row :  std_logic_vector(72 downto 0);


begin

-- Instantiate the Unit Under Test (UUT)
uut:compr PORT MAP(clk => clk,re_out_row=>re_out_row);

process
constant clkpule: time := 0.5 ns;
	begin
	clk<='1';
	wait for 0.5 ns;
	clk<='0';
	wait for 0.5 ns;
	end process;


process

--FILE outfile : TEXT IS OUT "re_out_row1.txt";
FILE reoutrowfile : TEXT IS OUT "re_out_rowfine18thjune12by13.txt";
 --variable vr_vcoef_out:std_logic_vector(40 downto 0);
 variable vclk:std_logic;
 variable line_reoutrow,line_out:line;
begin----------------------------------------------process of begin



--if(clk'event and clk='1') then
--while not(endfile(outfile)) loop
--clk<=vclk;
--r_ahvd_out<=vr_ahvd_out;
wait for 1 ns;

--WRITE( line_out,re_out_row);
		--WRITELINE( outfile, line_out);
```

```
            WRITE( line_reoutrow,re_out_row);
            WRITELINE( reoutrowfile, line_reoutrow);



            --wait for 10 ns;
            ---assert(ahvd_out=r_ahvd_out)
            ---report ("error: out mismatch") severity error;
            --end loop;
            --wait;
            --end if;
        END PROCESS;


end Behavioral;
```



In this CD total three folders are given and a DOCUMENT named file.

MATLAB Code:

1. Input image is given, that is input image is read from file.
2. Output is obtained in the form of image.
3. MATLAB code is given in first(1) folder

VHDL Code:

1. For this, input intensity matrix (Gray scale Image), threshold value are given as inputs.

2. Input given integer form are converted in to binary and further necessary steps are performed on binary.

3. Output is obtained in the form of coefficients in binary form unlike input given form.

4. This output coefficients which are in 2's complement representation (where out of total number of bits assigned, some bits indicate real, remaining bits indicate fractional part explained in chater 4) are converted into the image using MATLAB CODE given in third(3) folder to view the output image.
5. VHDL code is given in second (2) folder.