FPGA Implementation of DHT Algorithms for Image Compression

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

In

Electronics and communication engineering

By

Richa Agrawal
Roll no. 10609016

Department of Electronics and Communication Engineering
National Institute Of Technology, Rourkela
2009-2010

FPGA Implementation of DHT Algorithms for Image Compression

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

In

Electronics and communication engineering

By

Richa Agrawal
Roll no. 10609016

Under the supervision of

Dr. Kamala Kanta Mahapatra
Professor

Department of Electronics and Communication Engineering
National Institute Of Technology, Rourkela
2009-2010

ABSTRACT

Digital image processing is the use of computer algorithms to perform image processing on digital images. The basic operation performed by a simple digital camera is, to convert the light energy to electrical energy, then the energy is converted to digital format and a compression algorithm is used to reduce memory requirement for storing the image. This compression algorithm is frequently called for capturing and storing the images. This leads us to develop an efficient compression algorithm which will give the same result as that of the existing algorithms with low power consumption.

Compression is useful as it helps in reduction of the usage of expensive resources, such as memory (hard disks), or the transmission bandwidth required. But on the downside, compression techniques result in distortion (due to lossy compression schemes) and also additional computational resources are required for compression-decompression of the data. Reduction of these resources by comparing different algorithms for DHT is required.

FPGA Implementations of different algorithms for 1-DHT using VHDL as the synthesis tool are carried out and their comparison gives the optimum technique for compression. Finally 2-D DHT is implemented using the optimum 1-D technique for 8x8 matrix input. The results obtained are discussed and improvements are suggested to further optimize the design.

Contents

ACKNOWLEDGEMENT

I place on record and warmly acknowledge the continuous encouragement, invaluable supervision and inspired guidance offered by my guide **Dr. K. K. Mahapatra,** Professor, Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela, in bringing this report to a successful completion. This project has been a great learning experience and I am grateful to him for all his support and suggestions during this project.

I would also like to thank **Mr. Vijay Sharma**, M.tech student at NIT Rourkela, for his continuous encouragement and support during the completion of the project.

I am grateful to **Prof. S.K Patra**, Head of the Department of Electronics and Communication Engineering, for permitting me to make use of the facilities available in the department to carry out the project successfully. Last but not the least I express my sincere thanks to all of my friends who have patiently extended all sorts of help for accomplishing this undertaking.

Richa Agrawal

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Image compression, the art and science of reducing the amount of data required to represent an image, is one of the most useful and commercially successful technologies in the field of digital image processing. Digital image and video compression is now very essential. Internet teleconferencing, High Definition Television (HDTV), satellite communications and digital storage of movies would not be feasible unless a high degree of compression is achieved.

Compression is useful as it helps in reduction of the usage of expensive resources, such as memory (hard disks), or the transmission bandwidth required. In today's age of competition where everything is reducing its size every minute, the smaller is the better. But on the downside, compression techniques result in distortion (due to lossy compression schemes) and also additional computational resources are required for compression-decompression of the data.

1.1 Data compression

The term data compression refers to the process of reducing the amount of data required to represent a given quantity of information. Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain redundant data. Various techniques have been proposed for reducing the redundancy as far as possible.

Compression ratio is defined as the ratio of the size of compressed data to that of the uncompressed data.

So, $$C = \frac{size\ of\ compressed\ \ data}{size\ of\ uncompressed\ \ data} \qquad (1.1)$$

Redundancy is the reduction in size in comparison of the uncompressed size.

So, $$R = 1 - C \qquad (1.2)$$

Two-dimensional intensity arrays suffer from three principle data redundancies that can be identified and exploited:

- Coding redundancy

- Spatial and temporal redundancy
- Irrelevant information

1.2 Image compression model

In the first step of encoding process the image f(x,y) is mapped to a format to reduce spatial redundancy [2]. The various transforms used for mapping are

- Discrete cosine transform
- Discrete wavelet transform
- Discrete Hartley transform

Next quantization is done, where the loss of information takes place. Since it is an irreversible process, we can omit this step for a lossless coding technique.

The final step is symbol coding, where various coding techniques can be used to represent the information in minimum possible number of bits. The various coding techniques used are Huffman coding, run-length coding, LZW coding, bit plane coding, block transform coding and many other.

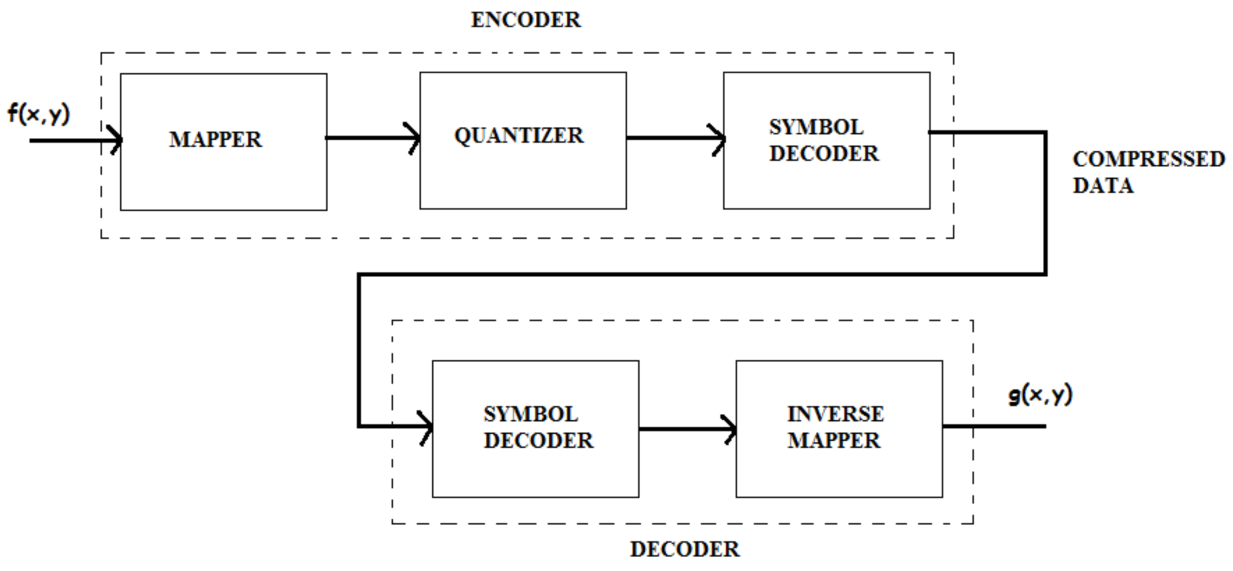


Figure 1.1 Functional block diagram of a general image compression system

1.3 DiscreteHartley Transform

The discrete Hartley transform is a linear, invertible function H: **R→R (**where **R** denotes the set of real numbers). The N real numbers$x_0 x_1 \ldots x_{N-1}$are transformed into N real numbers $H_0 H_1, \ldots . H_{N-1}$ according to the formula [8]:

$$H_k = \sum_{n=0}^{N-1} x_n \left(\cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N}\right) \text{for k= 0, 1… N-1} \qquad (1.3)$$

Properties:

1. The transform is a linear operator as it can be evaluated by the multiplication of the input series by an NxN matrix. Also the inverse transform can be evaluated by simply calculating the DHT of H_k multiplied by a factor 1/N.
2. The DHT can be used to compute both convolution and DFT.
3. It is a real valued function (unlike DFT) and the memory requirement to compute both forward and inverse DHT transforms is 50% that of the DCT.

Hence DHT is a better option for compression algorithms and is used for mapping the input image pixels and quantization.

Chapter 2

LITERATURE REVIEW

2.1 IC technology

Every processor must be implemented on an integrated circuit(IC). IC technology involves the manner in which we map a digital (gate level) implementation onto an IC. IC technologies differ by how customized the IC is for a particular design [3]. They are of three different types:

1. Full-custom/VLSI
2. Semi-custom/ASIC
3. Programmable logic device (FPGA)

In full custom IC technology, all layers for a particular embedded system's digital implementation are optimized. But this design has a very high non-recurring(NRE) cost and long turnaround time, typically many months. It is usually used only in high-volume or extremely performance-critical applications like in defense, spacecraft etc.

ASIC or Application specific integrated circuits are semi-custom ICs which can be implemented in two types: gate array and standard cell. In gate-array ASIC technology, the masks for the transistor level and gate levels are already built and in standard-cell ASIC technology, the masks for logic level cells such as NAND gate or AND-OR combinations are present. The designer has to connect the gates (routing) to implement the desired circuit. It has reduced NRE cost and faster time-to-market than full-custom designs.

An FPGA consists of arrays of field programmable logic blocks connected by programmable interconnected blocks. It is a more flexible and modular approach to PLD design. It is basically consists of look-up tables and flip flops. The FPGAs need to be programmed i.e. configuring the logic circuits and interconnection switches to implement a desired structural circuit. Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

2.2 FPGA Architecture

The FPGA is an integrated circuit that contains many large number of 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. Complex designs are created by combining these basic blocks to create the desired circuit.

Conceptually it can be considered as an array ofConfigurable Logic Blocks (CLBs) that can be connected together through a vast interconnectionmatrix to form complex digital circuits.

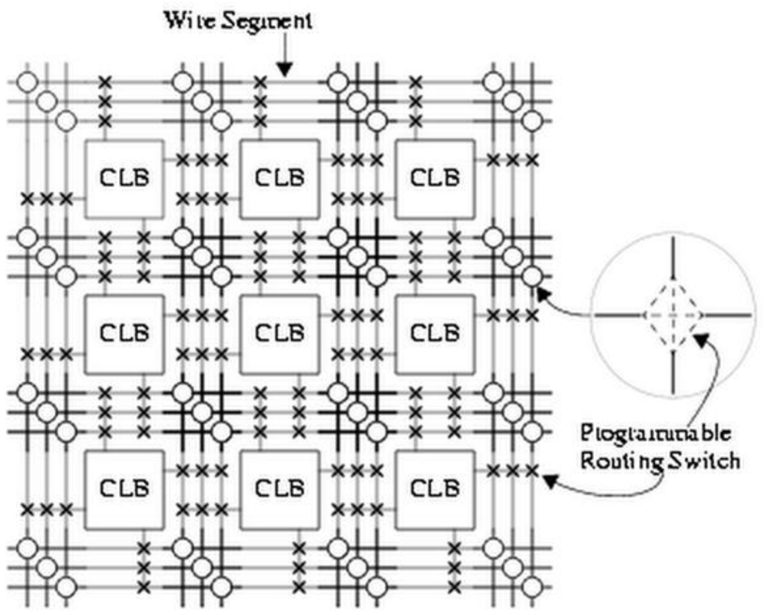


Figure 2.1: Basic Architecture of FPGA

The logic cell architecture varies between different device families.Generally speaking, each logic cell combines a few binary inputs (typically between 3 and 10) to one or two outputs according to a boolean logic function specified in the user program. The cell's combinatorial logic may be physically implemented as a small look-up table memory (LUT) or as a set of multiplexers and gates. LUT devices tend to be a bit more flexible and provide more inputs per cell than multiplexer cells at the expense of propagation delay.Programmable interconnectsprovide routing paths to connect the inputsand outputs of the logic cell and I/O blocks.

2.3 Image Compression

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time and bandwidthrequired for images to be sent over the Internet or downloaded from Web pages.

There are several different ways in which image files can be compressed. For Internet use, the two most common compressed graphic image formats are the JPEG format and the GIF format. The JPEG method is more often used for photographs, while the GIF method is commonly used for line art and other images in which geometric shapes are relatively simple.

The steps involved in image compression are as follows:

1. First of all the image is divided into blocks of 8x8 pixel values. These blocks are then fed to the encoder from where we obtain the compressed image.
2. The next step is mapping of the pixel intensity value to another domain. The mapper transforms images into a (usually non-visual) format designed to reduce spatial and temporal redundancy. It can be done by applying various transforms to the images. Here discrete Hartley transform is applied to the 8x8 blocks.
3. Quantizing the transformed coefficients results in the loss of irrelevant information for the specified purpose.
4. Source coding is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use, through use ofspecific encoding schemes.

The block diagram of the steps is given in figure 2.2

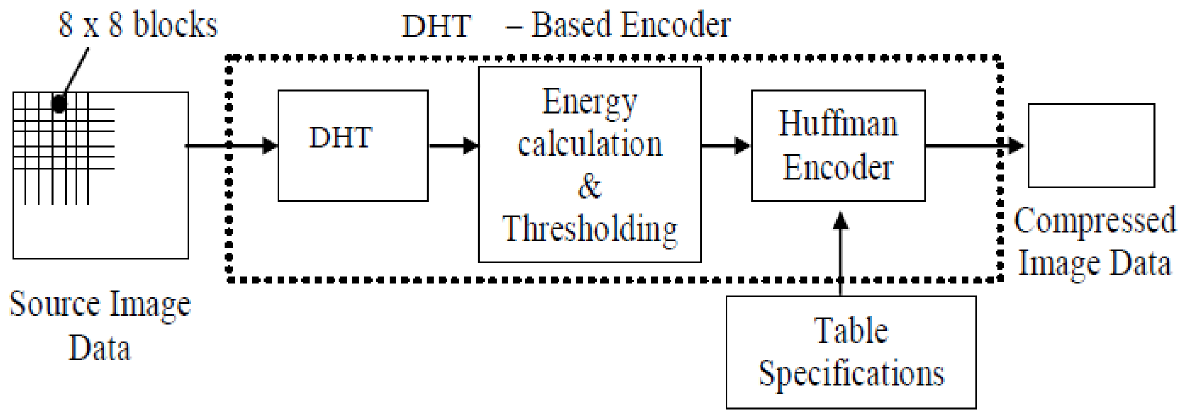


Figure 2.2: Energy quantization based image compression encoder

For retrieving the image back, the steps have to be reversed from the forward process. First the data is decoded using the decoder. Next inverse transform (IDHT) is calculated to get the 8x8 blocks. These blocks are then connected to form the final image. From the reconstructed imagepixel values it is clear that some of the high frequency components are preserved.This indicates that the edge property of the image is preserved.

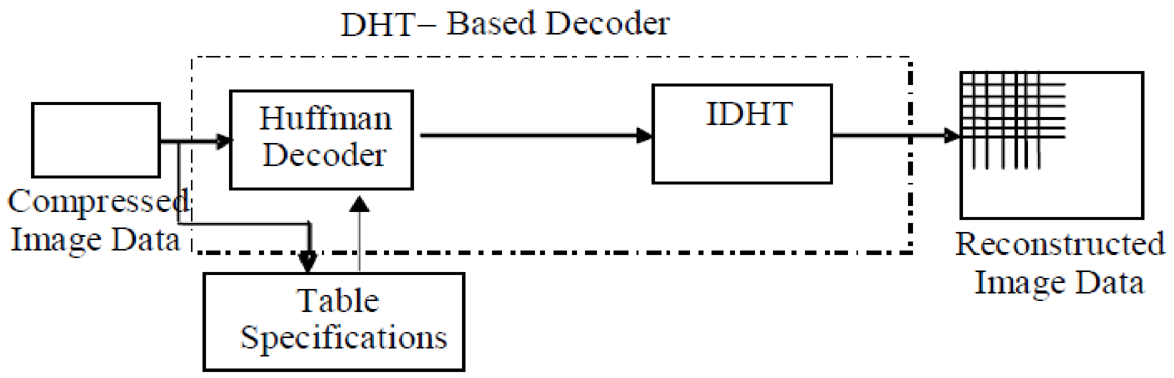


Figure 2.3: Energy quantization based image compression decoder

Different steps in image compression are as follows[1]:

2.3.1Transformation of image data

It is required to convert the pixel values into another domain so that it is easier to compress. A transform operates on an image's pixel values and converts them to a set of less correlated transformed coefficients. Natural images (which are the most common images to be compressed) have a lot of spatial correlation between the pixel intensities in its neighborhood. These correlations can be exploited by using the transform and so the spatial and temporal redundancy is reduced. This operation is generally reversible and may or may not reduce the data content of the images. Here discrete Hartley transform (DHT) is used for generating the coefficients.

2.3.2 Quantization

Quantization is the process of approximating a continuous range of values (or a very large set of possible discrete values) by a relatively small ("finite") set of discrete symbols or values. In other words it means mapping a broad range of input values to a limited number of output values.It reduces the accuracy of the transformed coefficients in accordance with a pre-established fidelity criterion. The goal is to reduce the amount of irrelevant information present in the image. Since information is lost in this process, it is an irreversible process. In error-free techniques this step hence must be omitted to keep the whole information intact.

The human eye is fairly good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This fact allows one to get away with a greatly reduced amount of information in the high frequency components. This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer. This is the main lossy operation in the whole process. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers. The quantization matrices are formed for different transforms according to their frequency distribution in the coefficient matrix.

Quantization matrix for DCT can be easily obtained but it is difficult for DHT since the scanning order is special for DHT. The scanning order for DHT is given in figure 2.4.

1	2	4	6	8	10	12	14
3	16	17	19	21	23	25	27
5	18	29	30	32	34	36	38
7	20	31	40	41	43	45	47
9	22	33	42	49	50	52	54
11	24	35	44	51	56	57	59
13	26	37	46	53	58	61	62
15	28	39	48	55	60	63	64

Figure2.4: Scanning order for DHT

Since it is difficult to design the quantization matrix, energy quantization method can be applied. In this method the energy content of each matrix of transformed coefficients is obtained by the following formula. The normalized energy is given by:

$$E_n = \sum_{m=0}^{M} \sum_{n=0}^{N} x(m,n)^2 \tag{2.1}$$

where M and N are the widths of the sample block and x(m,n) is the transformed sample.

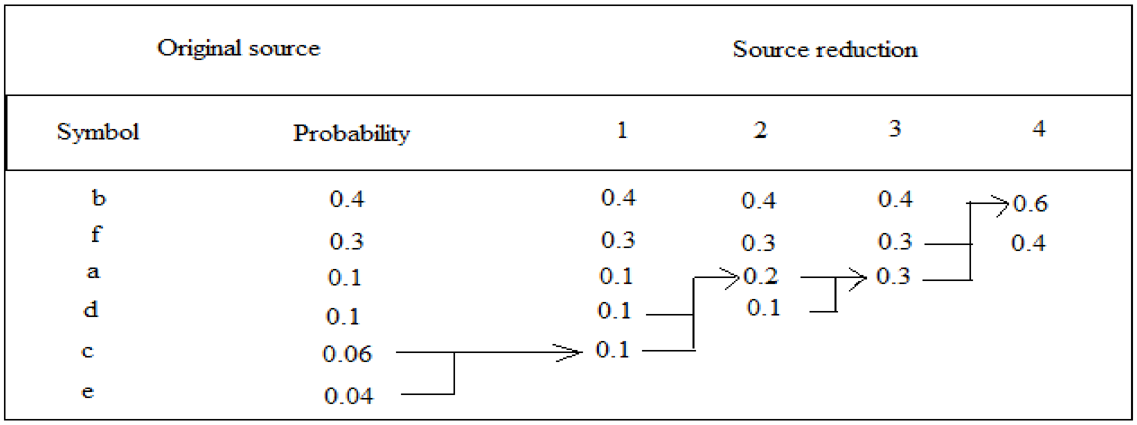
Next a threshold value is selected (i.e. pre-defined according to the fidelity criterion) according to which the transformed values will be truncated or kept intact. The threshold value is not a global value but determined as a percentage of the energy content of the matrix and hence varies for each matrix. The percentage value is only pre-decided. If the transformed co-efficient is less than the threshold value, it is truncated otherwise kept intact. This helps in treating the image in segments and sustaining the information in different regions of the images. For higher compression rates the threshold value is increased and for lower compression the threshold value is kept large (close to normalized value).

2.3.3 Entropy coding

An entropy encoding is a lossless data compression scheme that is independent of the specific characteristics of the medium.One of the main types of entropy coding creates and assigns a unique prefix code to each unique symbol that occurs in the input. These entropy encoders then compress data by replacing each fixed-length input symbol by the corresponding variable-length prefix code word. The length of each code word is approximately proportional to the negative logarithm of the probability. Therefore, the most common symbols use the shortest codes.Two of the most common entropy encoding techniques are Huffman coding and arithmetic coding.

2.3.3.1Huffman coding: it is one of the most popular techniques for removing coding redundancy. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.A Huffman coder determines the compressed symbols by forming a data treefrom the original data symbols and their associated probabilities.

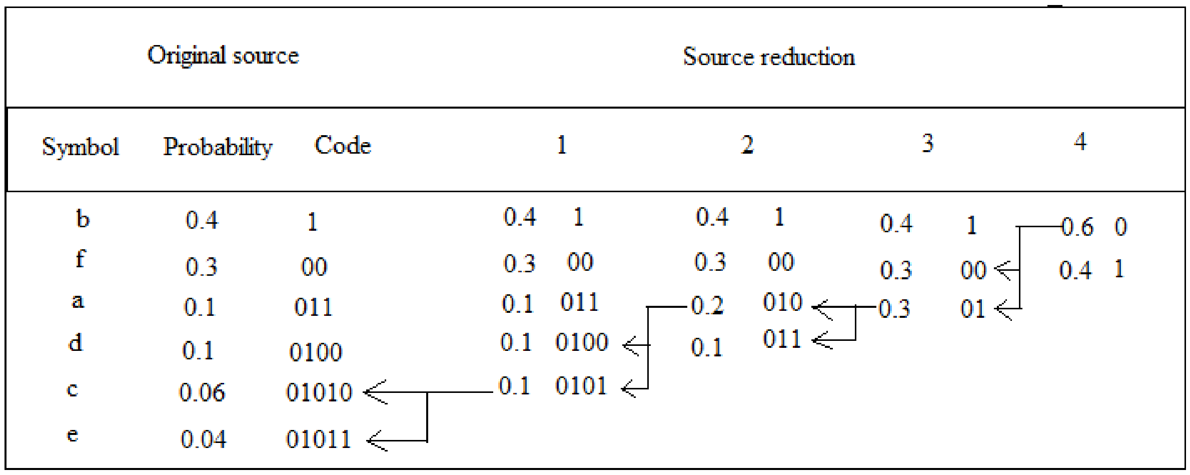
The first step in Huffman coding is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. It is shown in figure 2.5 as an example.



Original source		Source reduction			
Symbol	Probability	1	2	3	4
b	0.4	0.4	0.4	0.4	0.6
f	0.3	0.3	0.3	0.3	0.4
a	0.1	0.1	0.2	0.3	
d	0.1	0.1	0.1		
c	0.06	0.1			
e	0.04				

Figure 2.5: Huffman source reductions

The second step is to code each reduced source, starting with the smallest source and working back to the original source, as shown in figure 2.6. The minimal length binary code for a two-symbol source is the symbols 0 and 1.



Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
b	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
f	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a	0.1	011	0.1	011	0.2	010	0.3	01		
d	0.1	0100	0.1	0100	0.1	011				
c	0.06	01010	0.1	0101						
e	0.04	01011								

Figure 2.6: Huffman code assignment procedure

Huffman’s procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols can be coded one at a time.

2.3.3.2 Run-length coding: Run-length is the number of bits for which signal remains unchanged. A run-length of 3 for bit 1, represents a sequence of '111'. Images with repeating intensities along their rows (or columns) can often be compressed by representing runs of identical intensities a run-length pairs, where each run length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity.

2.4 Discrete Hartley Transform

The Hartley transform is an integral transform closely related to the Fourier transform, but which transforms real-valued functions to real-valued functions. It was proposed as an alternative to the Fourier transform by R. V. L. Hartley in 1942[8]. Compared to the Fourier transform, the Hartley transform has the advantages of transforming real functions to real functions (as opposed to requiring complex numbers) and of being its own inverse.The discrete version of the transform, the Discrete Hartley transform, was introduced by R. N. Bracewell in 1983.

2.4.1 Formula

Formally, the discrete Hartley transform is a linear, invertible function H: **R→R (**where **R** denotes the set of real numbers). The N real numbers$x_0 x_1 \ldots x_{N-1}$ are transformed into N real numbers$H_0 H_1 \ldots H_{N-1}$ according to the formula[6]:

$$H_k = \sum_{n=0}^{N-1} x_n \left(\cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N}\right) \text{for k= 0, 1… N-1} \qquad (2.2)$$

The inverse transform is given by:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} H_k \left(\cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N}\right) \text{for n= 0, 1… N-1} \qquad (2.3)$$

The casfunction is given by:

$$cas(\frac{2\pi nk}{N}) = \cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N} \qquad (2.4)$$

and one of the properties of *cas* function is:

$$2cas(a+b) = cas(a)cas(b) + cas(-a)cas(b) + cas(a)cas(-b) - cas(-a)cas(-b) \quad (2.5)$$

2 –Dimensional DHT of an array x (m, n) of size MxN may be defined as:

$$X(k,l) = \sum_{m=0}^{M}\sum_{n=0}^{N} x(m,n)cas(\frac{2\pi mk}{M} + \frac{2\pi nl}{N})$$

for k=0,1….M-1 & l=0,1…..N-1 (2.5)

The inverse transform is given by the same formula along with a scaling factor of 1/MN i.e.

$$X(k,l) = \frac{1}{MN}\sum_{m=0}^{M}\sum_{n=0}^{N} x(m,n)cas(\frac{2\pi mk}{M} + \frac{2\pi nl}{N})$$

for k=0,1….M-1 & l=0,1…..N-1 (2.6)

2.4.2 Fourier Transform and Convolution

The real and imaginary parts of the Fourier transform are given by the even and odd parts of the Hartley transform, respectively

$$F(w) = \frac{H(w) + H(-w)}{2} - \frac{i\left(H(w) - H(-w)\right)}{2} \qquad (2.7)$$

There is also an analogue of the convolution theorem for the Hartley transform. If two functions $x(t)$ and $y(t)$ have Hartley transforms $X(\omega)$ and $Y(\omega)$, respectively, then their convolution $z(t) = x * y$ has the Hartley transform:

$$Z(w) = \{H(x * y)\} = \frac{\sqrt{2pi}\,(X(w)[Y(w) + Y(-w)] + X(-w)[Y(w) - Y(-w)])}{2} \qquad (2.8)$$

Similar to the Fourier transform, the Hartley transform of an even/odd function is even/odd, respectively.

2.4.3 Properties:

1. The transform is a linear operator as it can be evaluated by the multiplication of the input series by an NxN matrix. Also the inverse transform can be evaluated by simply calculating the DHT of H_k multiplied by a factor 1/N.
2. The DHT can be used to compute both convolution and DFT.
3. It is a real valued function (unlike DFT) and the memory requirement to compute both forward and inverse DHT transforms is 50% that of the DCT.

Hence DHT is a better option for compression algorithms and is used for mapping the input image pixels and quantization.

2.5 Performance Measures of Image Compression

Normally the performance of a data compression scheme can be measured in termsof three parameters. These are:

1. Compression efficiency: Compression efficiency is measured through compression ratio (CR). The compression ratio can be defined as the ratio of the data size (number of bits) of the original data to thesize of the corresponding compressed data. After the image has been compressed, the memory requirement for storage reduces. CR gives the measure of this reduction in storing images.

2. Complexities: The complexities of a digital datacompression algorithms are measured by a number of data operations requiredperforming both the encoding and decoding process. The data operations includeadditions, subtractions, and multiplication, divisions and shift operations.
3. Distortion measurement for lossy compression: In the lossy compression algorithms, distortion measurement is used to measure the amount of information lost after reconstructing the original signal or image data that has been recovered from the compressed data through encoding and decoding operations. The mean square error (MSE) is one of the distortion measurements in the reconstructed data. The performance measurement parameter; signal to noise ratio (SNR) is also used to measure the performance of thelossy compression algorithms.

   Mean square error for a 1-D data is given by:

$$MSE = \frac{1}{N}\sum_{n=0}^{N-1}|x(n) - x^{'}(n)|^2 \qquad (2.9)$$

   where N is the number of pixels in the image, *x*(*n*) is the original data and *x*'(*n*) is the compressed data.

   Peak Signal to Noise ratio (PSNR) is given by:

$$PSNR = 10log_{10}\left(\frac{255^2}{MSE'}\right) \qquad (2.10)$$

   Where MSE' is calculated for 2-D block as:

$$MSE^{'} = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}|x(m,n) - x^{'}(m,n)|^2 \qquad (2.11)$$

Chapter 3

PROBLEM STATEMENT

FPGAs - Field Programmable Gate Arrays - are future-oriented building bricks which allow perfect customization of the hardware at an attractive price even in low quantities. FPGA components available today have usable sizes at an acceptable price. This makes them effective factors for cost savings and time-to-market when making individual configurations of standard products.A time consuming and expensive redesign of a board can often be avoided through application-specific integration of IP cores in the FPGA - an alternative for the future, especially for very specialized applications with only small or medium volumes.

3.1 DHT vs. DCT

Many papers have been published describing various algorithms for implementation of 2-D DHT in hardware. Discrete Hartley Transform is the real valued transform which gives only real transform coefficients for real input stream. It has the main advantage over DCT (Discrete Cosine Transform, which is the most common technique now) of reducing the memory content up to 50% since the inverse transform is identical to the forward transform. Also, it retains the higher frequency components, which restores the detailing (such as sharp boundaries) of the image. Since it is a real valued function unlike DFT, the computational complexities are also lower than in DFT algorithms.

3.2 Advantages of FPGAs

FPGAs have mostly become more popular in the past three years. It is a reprogrammable logic device and can be configured by the end-user (field programmable) to have specific circuitry within it. The main advantages of FPGA over other design technologies are listed below:

- <u>Fast prototyping and turnaround</u> – Prototyping means building an actual circuit to a theoretical design to verify that it works, and to provide a physical platform for debugging it if it does not.Turnaround is the total time between submission of a process and its completion. Since in FPGAs all the interconnects are already present and the designer only has to fuse these programmable interconnects to get the desired logic output, the time taken is quite less compared to ASIC or full-custom design. It is programmed by users at their site usingprogramming hardware. Today all the leading companies are able to launch new products every other month due to this advantage of FPGAs only.

- NRE cost is zero- Non-recurring engineering (NRE) refers to the one-time cost of researching, developing, designing, and testing a new product. Since FPGAs are reprogrammable and they can be used without any loss of quality every time, the non-recurring cost is not present. This greatly decreases the initial cost of manufacturing of ICs since the programs can be run and tested on the FPGAs free of cost.
- High speed-Since the FPGA technology is based on look-up tables, the time taken to execute is less than that in ASIC technology. This high speed is used in making various multipliers today, which had traditionally been the sole reserve of DSP processors.
- Parallel processing- FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute-force attack, of cryptographic algorithms. The inherent parallelism of the logic resources on an FPGA allows for considerable computational throughput even at a low MHz clock rates. The flexibility of the FPGA allows for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units.
- Low cost-The cost of FPGA is quite affordable and hence it makes them very designer-friendly. Also the power requirement is less since the architecture of FPGAs is based on LUTs.

Due to the above mentioned advantages of FPGAs in IC technology and DHT in mapping of images, implementation of 2-D DHT in FPGA can give us a clearer idea about the advantages and limitations of using DHT as the mapping function. It can surpass the now most common compressed graphic image formats using DCT and can help in forming better image processing and restoration techniques.

FPGA implementation of the design is done using VHDL as the synthesis tool. The package details of the FPGA and simulator used are listed below:

1. Family: Virtex II Pro
2. Device: XC2VP30
3. Package: FF896
4. Speed grade: -7
5. Synthesis tool: XST (VHDL/Verilog)
6. Simulator: (i) Model Sim 6.2C

   (ii) ISE Simulator

Chapter 4

DIFFERENT MODELING TECHNIQUES AND ARCHITECTURES DEVELOPED

The DHT belongs to the family of frequencytransforms that map temporal or spatial functionsinto frequency functions. .The DHT accomplishesthis in amanner similar to the better-known FourierTransform. The significant difference between theDiscrete Fourier Transform (DFT) and DHT'salternative is that the DHT usesonly real values,i.e., no complex numbers.

The DHT achieves this via the kernel or ***cas***function:

$$cas\left(\frac{2\pi nk}{N}\right) = \cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N} \qquad (4.1)$$

The N-point (DHT) is given by the followingformula[8]:

$$Y_k = \sum_{n=0}^{N-1} X_n \left(\text{cas}\,\frac{2\pi nk}{N}\right) \text{for k= 0, 1… N-1} \qquad (4.2)$$

where, $\text{H}_{\text{nk}} = \text{cas}(\frac{2\pi nk}{N})$, is the transform kernel.

Two architectures have been implemented for computing DHT and their efficiencies studied regarding FPGA implementation. They are systolic architecture and distributed arithmetic architecture.

4.1 Baugh-Wooley algorithm for multiplication

It is an algorithm for high-speed, two’s complement, m-bit by n-bit parallel multiplication. The two’s complement multiplication is converted to an equivalent parallel array addition problem in which each partial product is the AND of a multiplier bit and a multiplicand bit, and the signs of all the partial product bits are positive [7].

The algorithm’s principle advantage is that the signs of all the partial products are positive, allowing the product to be formed using array addition techniques. Therefore the product is formed with only the AND function and the ADD function. No subtraction is necessary, nor is the NAND function needed. For 8x8 bit multiplier, the output is a 16-bit binary number.

The Baugh Wooley multiplier is hence used due to its simplicity, regularity and high through-put rate which can be achieved for any transform size and word-length of the input data. It is implemented in VHDL using full-adders and in-built AND functions.

4.2 DHT based Systolic Architecture (SA)

4.2.1 Mathematical modeling

If the elements of the transform's kernel and the input vector are represented using the 2's complement number representation[4], then

$$H_{ik} = -h_{ik,n-1}2^{n-1} + \sum_{l=0}^{n-2} h_{ik,l}2^{l} \tag{4.3}$$

And,

$$X_{k} = -x_{k,n-1}2^{n-1} + \sum_{m=0}^{n-2} x_{k,m}2^{m} \tag{4.4}$$

Where $h_{ik,l}$ and $x_{k,m}$ are the l-th bit ofH_{ik} and m-th bit of X_k respectively and $h_{ik,n-1}$ and $x_{k,n-1}$are sign bits, where n is the word length.

So, the transform coefficient Y_i can becomputed as follows:

$$Y_i = \sum_{k=0}^{N-1}\left[-h_{ik,n-1}2^{n-1} + \sum_{l=0}^{n-2} h_{ik,l}2^{l}\right]\left[-x_{k,n-1}2^{n-1} + \sum_{m=0}^{n-2} x_{k,m}2^{m}\right] \tag{4.5}$$

From the above equation it can be seen that the computation of the matrix product depends on the type of multiplier used. So, Baugh-Wooley multiplier algorithm is used. Hence the equation obtained is:

$$Y_i = \sum_{k=0}^{N-1}\left[\sum_{l=0}^{n-2}\sum_{m=0}^{n-2} 2^{l+m} h_{ik,l}x_{k,m} + 2^{2n-2} h_{ik,n-1}x_{k,n-l} + \left(\sum_{l=0}^{n-2} -2^{l}h_{ik,l}x_{k,n-1} + \sum_{m=0}^{n-2} -2^{m}x_{k,m}h_{ik,n-1}\right)2^{n-1}\right] \tag{4.6}$$

The above equation can be mapped into the architecture, as shown in the figure 4.1 for 4-point DHT i.e. N=4.

4.2.2 Architecture

The architecture for 4-point DHT is shown in figure 4.1 . It consists of 16 identical processing elements (PE's)[4].

Each PE consists of a parallel Baugh-Wooley multiplier, storage elements where the coefficients h_{ik} and x_{kj} are stored in a storage element for pipelining the partial products and a parallel adder based on fast carry is used to add the result of the partial product by the previous one.

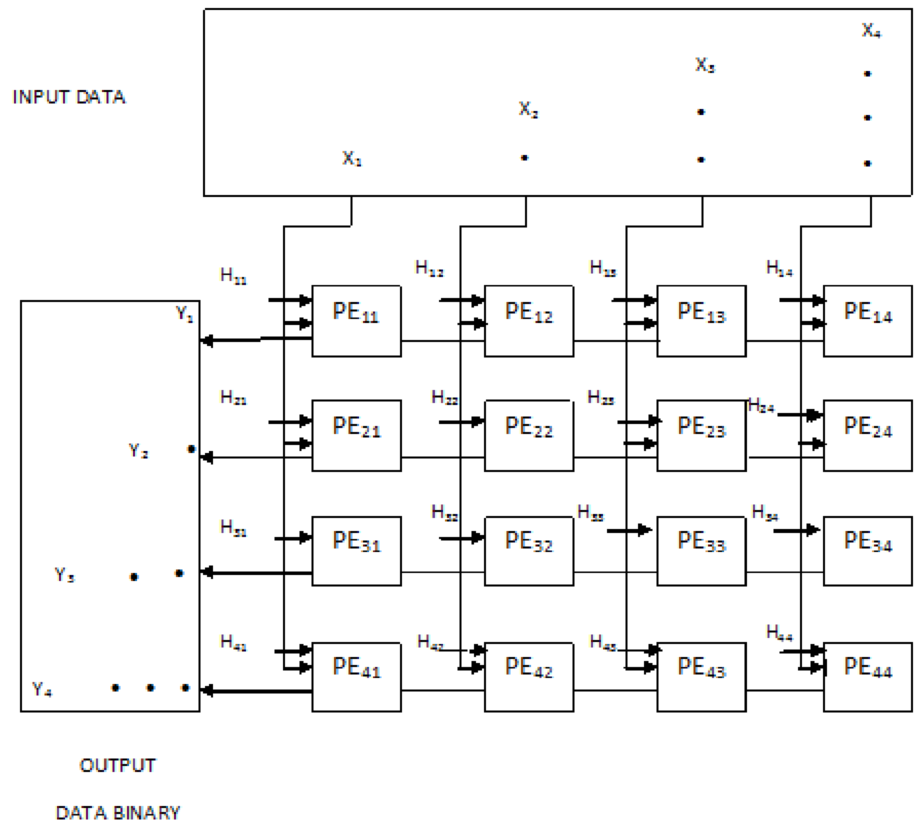


Figure 4.1: Systolic architecture for DHTs (N=4)

The input data elements X_j are fed from the north in a parallel fashion while the kernel matrix elements fixed in their corresponding PE cells (during the entire calculation) are fed parallel too.

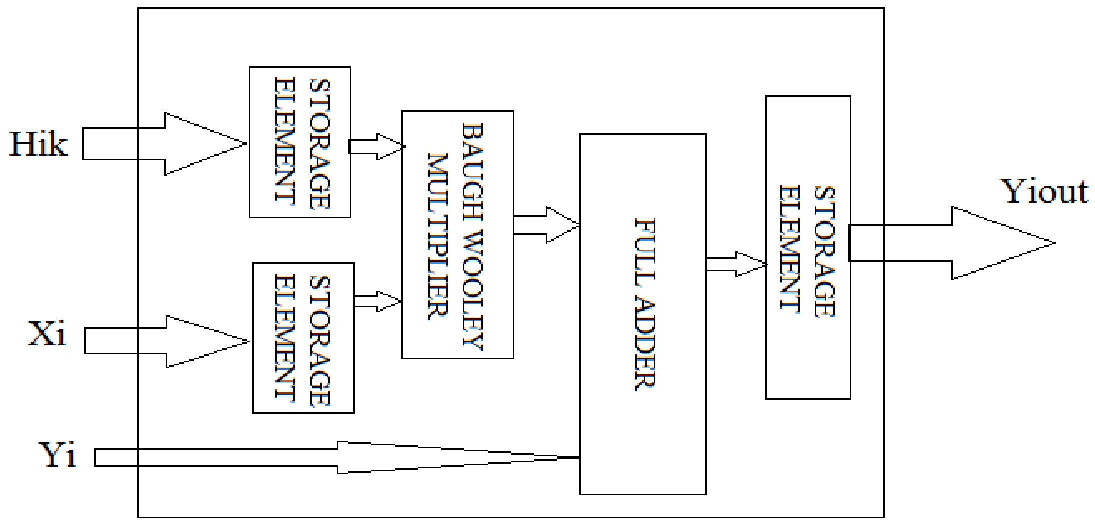


Figure 4.2: Structure of a Processing element

The structure of each processing element is given in figure 4.2

4.3 DHT based Distributed Arithmetic design methodology (DA)

4.3.1 Mathematical modeling

This approach is based on distributed arithmetic Read Only Memory (ROM), accumulator structure and offset binary coding (OBC) techniques. The OBC technique reduces the ROM size by a factor of 2 to 2^{N-1} when using DA principles. It is a technique where all-zero corresponds to the minimal negative value and all-one to the maximal positive value.

Suppose that $\{H_{ik}\}$'s are L-bits constants and $\{X_k\}$'s are written in the fractional format as shown[4]:

$$X_k = -x_{k,n-1} + \sum_{m=1}^{n-1} x_{k,n-1-m} 2^{-m} \tag{4.7}$$

Now, rewriting equation 4.7, we get

$$X_k = \frac{[X_k - (-X_k)]}{2} \tag{4.8}$$

or, $X_k = [-(x_{k,n-1} - \overline{x_{k,n-1}}) + \sum_{m=1}^{n-1}(x_{k,n-1-m} - \overline{x_{k,n-1-m}})2^{-m} - 2^{-(n-1)}]/2$ (4.9)

where

$$-X_k = \overline{-x_{k,n-1}} + \sum_{m=1}^{n-1} \overline{x_{k,n-1-m}}\ 2^{-m} + 2^{-(n-1)} \tag{4.10}$$

Now we define,

$$d_{kj,m} = \{\, x_{k,m} - x_{k,m}\ \ , for\ m \neq n-1\ \ and\ \ -(x_{k,n-1} - x_{k,n-1})\, , for\ m = n-1) \tag{4.11}$$

And $d_{k,m} \in \{-1,+1\}$, so equation 4.10 can be rewritten as:

$$X_k = \frac{\left[\sum_{m=0}^{n-1} d_{k,n-1-m}\, 2^{-m} - 2^{-(n-1)}\right]}{2} \tag{4.12}$$

Now using the above equation 4.12, we calculate DHT

$$Y_i = \sum_{k=0}^{N-1} H_{ik}/2\ [\sum_{m=0}^{n-1} d_{k,n-1-m}\, 2^{-m} - 2^{-(n-1)}] \tag{4.13}$$

$$Y_i = \sum_{m=0}^{n-1} (\sum_{k=0}^{N-1} \frac{H_{ik}\, d_{k,n-1-m}}{2}) 2^{-m} - \left(\frac{\sum_{k=0}^{N-1} H_{ik}}{2}\right) 2^{(n-1)} \tag{4.14}$$

Now we define,

$$D_{im} = \sum_{k=0}^{N-1} (1/2) H_{ik}\, d_{k,m} \qquad , 0 \leq m \leq W-1 \tag{4.15}$$

And

$$D_{iextra} = -1/2 \sum_{k=0}^{N-1} H_{ik} \tag{4.16}$$

.

Therefore Y_i can be computed as :

$$Y_i = \sum_{m=0}^{n-1} D_{i,n-m-1} 2^{-m} + D_{iextra}\, 2^{-(n-1)} \tag{4.17}$$

So, for N=3 the contents of the ROM will reduce from 8 to 4 values as shown in the table. Here x_1, x_2, x_3 are the input bit vectors and m denotes the position of the bit.

Table 4.1: The contents of ROM i

$x_{1,m}$	$x_{2,m}$	$x_{3,m}$	The contents of ROMi
0	0	0	$-\frac{H_{i1}+H_{i2}+H_{i3}}{2}$
0	0	1	$-\frac{H_{i1}+H_{i2}-H_{i3}}{2}$
0	1	0	$-\frac{H_{i1}-H_{i2}+H_{i3}}{2}$
0	1	1	$-\frac{H_{i1}-H_{i2}-H_{i3}}{2}$
1	0	0	$-\frac{-H_{i1}+H_{i2}+H_{i3}}{2}$
1	0	1	$-\frac{-H_{i1}+H_{i2}-H_{i3}}{2}$
1	1	0	$-\frac{-H_{i1}-H_{i2}+H_{i3}}{2}$
1	1	1	$-\frac{-H_{i1}-H_{i2}-H_{i3}}{2}$

Since the last four rows are identical to the first four except for the first bit, they can removed and only four ROMs can be sufficient for the calculation. So, the new contents of the ROM are as shown in table.

Table 4.2 The new contents of ROM i

$x_{1,m}$	$x_{2,m}$	$x_{3,m}$	The contents of ROMi
0	0	0	$-\frac{H_{i1}+H_{i2}+H_{i3}}{2}$
0	0	1	$-\frac{H_{i1}+H_{i2}-H_{i3}}{2}$
0	1	0	$-\frac{H_{i1}-H_{i2}+H_{i3}}{2}$
0	1	1	$-\frac{H_{i1}-H_{i2}-H_{i3}}{2}$

4.3.2 Architecture

The figure below shows the architecture for the computation of DHTs (N=3) using DA principles with OBC scheme[4]. The computation starts from LSB of x i.e. m=0.

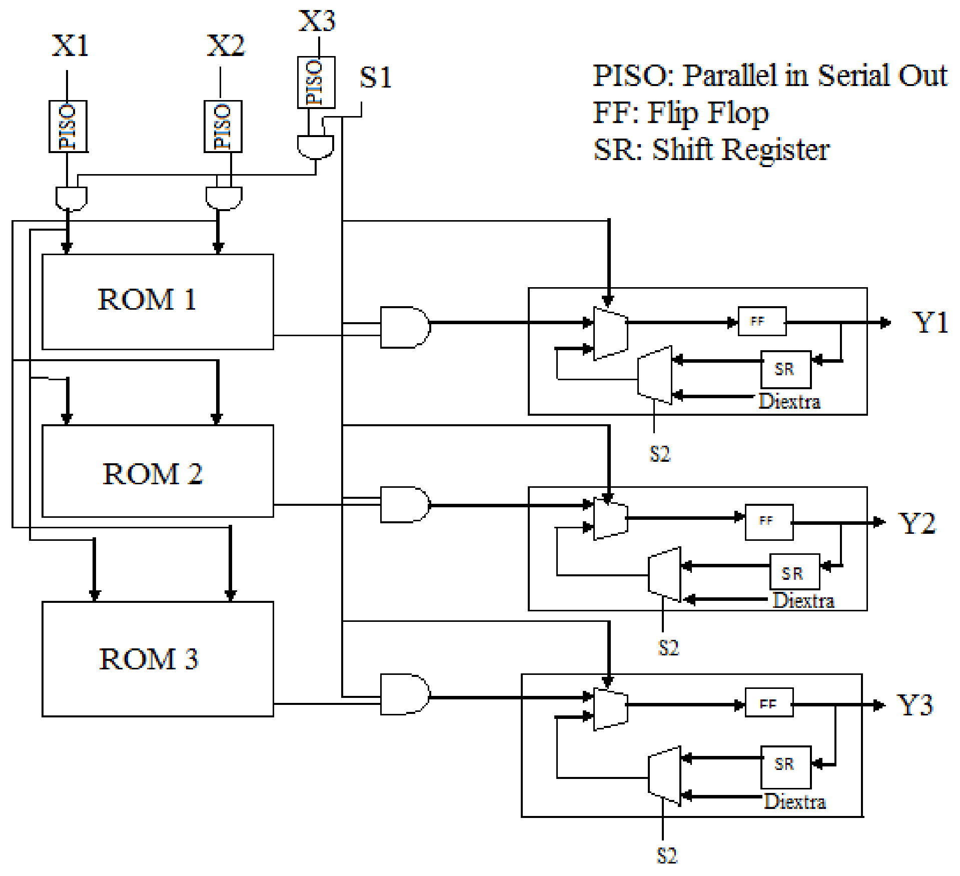


Figure 4.3: DHT based OBC using DA principles

First the input data enters the PISO (Parallel In Serial Out), so that the bits of the input data vector comes out serially starting with their LSBs. The XOR gates are used for address decoding, i.e. only 2 bits are required to locate the memory location in ROM. So 1st and 2ndbit are XORed with the 3rd bit to get the memory location. Also the third bit is used to determine whether addition or subtraction will take place during accumulation. PISO consists of a clock signal, input data vector and the single bit output which gives the bits of the vector input serially. Same PISO can be used for all the input vectors, and they should work parallel at the same time.

ROM is a memory which stores the constants used in the distributed arithmetic method from the table 2. It consists of registers which is in the form of an array so that the contents can be exactly located, like in the memory.The contents of each ROM are different, so for a three input (N=3), three ROMs are required. Similarly for an N-input DHT, 'N' ROMs will be required. The table gives the contents of the ith ROM. Each ROM will contain $2^{(N-1)}$ constants instead of 2^N constants due to their repeatability.

The 'Shift and Accumulate' block gives the output after addition/subtraction of the ROM contents. Initially the contents of the accumulator are reset. After each clock cycle, the accumulator is shifted to the left and the ROM output is added/subtracted according to the 3rd input bit. Finally after the last shift, the term D_{iextra} added to the accumulator. This gives the final transformed output for the ith input. For N-input DHT, N number of identical 'shift-accumulate' blocks are required, and the N-point DHT outputs are derived from them. For an N-point DHT, (N+2) clock cycles are required to obtain the output.

4.4 Eight point DHT with pipelined stages with delays

4.4.1 Mathematical modeling

The DHT of a real-valued-point input vector,$x_0 x_1 \ldots x_{N-1}$, may be defined as

$$X_k = \sum_{n=0}^{N-1} \boldsymbol{x_n} C_N(k,n) \qquad (4.18)$$

Where

$$C_N(k,n) = cas(\frac{2\pi nk}{N}) = \cos\frac{2\pi nk}{N} + \sin\frac{2\pi nk}{N} \quad \text{for k,n=0,1....N-1} \qquad (4.19)$$

Supposing N to be an even number, the sequence $\boldsymbol{x_n}$ is divided into two sub-sequences$\boldsymbol{x_1}$ and$\boldsymbol{x_2}$length N/2 each, such that $\boldsymbol{x_1} = \{x_0, x_2 \ldots\ldots x_{N-2}\}$contains all even – indexed termsand $\boldsymbol{x_2} = \{x_1, x_3 \ldots\ldots x_{N-1}\}$ contains all odd-indexed terms of the input sequence $\boldsymbol{x}$.

Then the DHT can be defined as[5] :

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} \boldsymbol{x_1} C_N(k, 2n) + \sum_{n=0}^{\frac{N}{2}-1} \boldsymbol{x_2} C_N(k, 2n+1) \tag{4.20}$$

Let X_{1k}andX_{2k} represent the (N/2) –point DHT coefficients of sequences $\boldsymbol{x_1}$ and$\boldsymbol{x_2}$of length (N/2) respectively. Using the symmetry properties of sine and cosine functions, the N-point DHT may be expressed as the following set of equations:

$$X_k = X_{1k} + E_k \tag{4.21}$$

$$X_{M+k} = X_{1k} - E_k \tag{4.22}$$

$$E_k = X_{2k} \cos\left(\frac{\pi k}{M}\right) + X_{2(M-k)}\, sin\left(\frac{\pi k}{M}\right) \tag{4.23}$$

For k=1,2….M-1 where M=N/2.

4.4.2 Architecture

Hence for computing 8-point DHT from 4-point DHT the set of equations obtained is given by equation 4.24[5]:

1. $X_0 = X_{10} + X_{20}$
2. $X_1 = X_{11} + (X_{21} + X_{23})/\sqrt{2}$
3. $X_2 = X_{12} + X_{22}$
4. $X_3 = X_{13} + (X_{21} - X_{23})/\sqrt{2}$ (4.24)
5. $X_4 = X_{10} - X_{20}$
6. $X_5 = X_{11} - (X_{21} + X_{23})/\sqrt{2}$
7. $X_6 = X_{12} - X_{22}$
8. $X_7 = X_{13} - (X_{21} - X_{23})/\sqrt{2}$

So, for computing 8-point DHT the multiplication with $1/\sqrt{2}$ can be read from a ROM, while a block of pipelined adders perform the addition.

It computes DHT in 5 pipelined stages. For first two stages, it consists of two 4-pointDHT modules that receive the odd and even indexed subsequences$\boldsymbol{x_1}$ and$\boldsymbol{x_2}$and from the input buffer. In the third pipelined stage, multiplication with $1/\sqrt{2}$ is done for the required coefficients i.e. X_{21}andX_{23}. Next they are added and subtracted in the fourth stage. During 3rd and 4th stages the rest of the coefficients are passed through a delay. Delay consists of simply registers i.e. they are stored in different registers and passed to the next stage. Finally the fifth pipelined stage is a parallel adder block which adds/subtracts the coefficients to give the desire output.

The block diagram of the described method is given in figure 4.4.

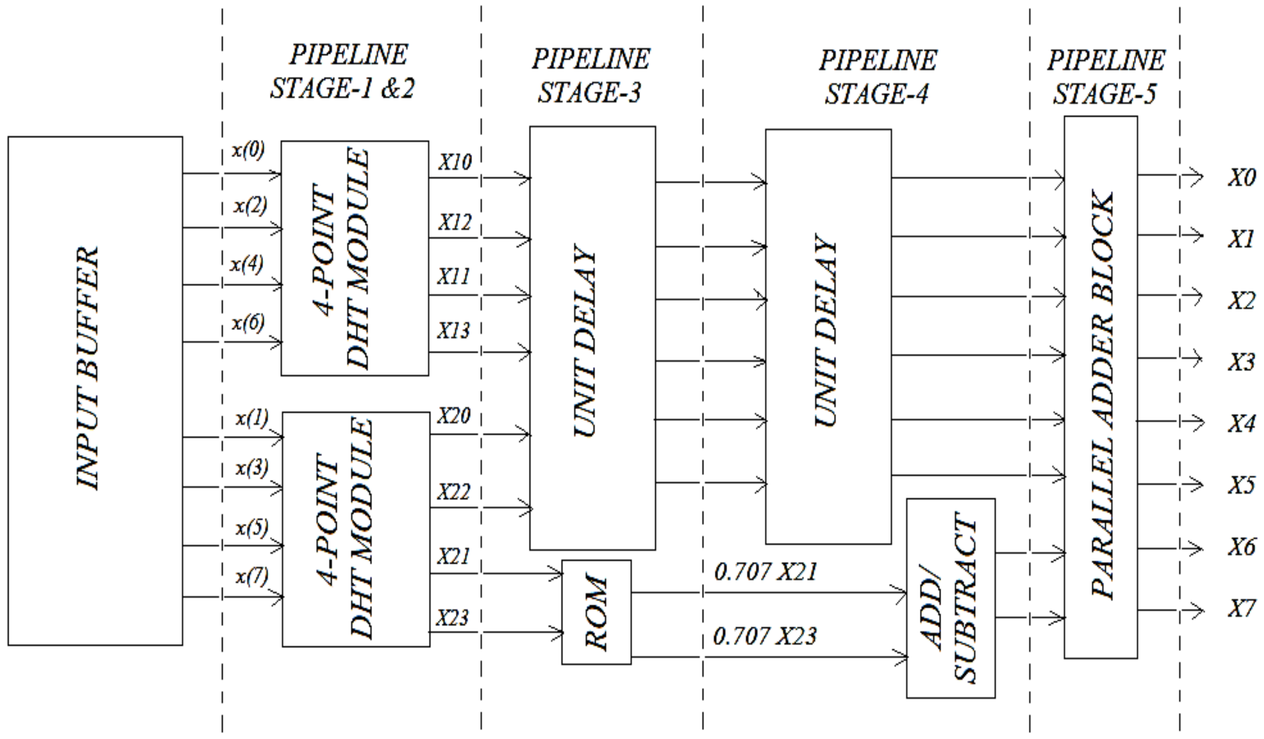


Figure 4.4: Flow chart of the 8-point DHT in pipelined approach with delays

4.5 Two-Dimensional DHT

4.5.1 Mathematical modeling

Two-dimensional DHT can be computed using the 1-D DHT blocks. Various methods have been proposed for this architecture. The one implemented follows the algorithm given below. Let the size of the input 2-D matrix **'F'** be 8x8, i.e. M=N=8 [6].

1. First 1-D DHT of all the rows of matrix **F** are taken and stored in another 8x8 matrix **'G'**.
2. Next 1-D DHT of all the columns obtained in the matrix **G**is computed and stored in another matrix **T.**

   The temporary outcome is of the form, which is not Hartley transform. It is given by:

$$\mathrm{T(u,v)} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} \mathrm{f(x,y)cas}(\frac{2\pi ux}{M})cas\left(\frac{2\pi vy}{N}\right) \tag{4.25}$$

3. However it can be converted to Hartley transform by using the trigonometric identity eq(2.5),

$$2cas(a+b) = cas(a)cas(b) + cas(-a)cas(b) + cas(a)cas(-b) - cas(-a)cas(-b).$$

Hence the desired Hartley transform can be expressed as the sum of four temporary transforms

$$2H(u,v) = T(u,v) + T(M-u,v) + T(u,N-v) - T(M-u,N-v) \tag{4.26}$$

here MxN is the size of the input matrix. Therefor M=N=8. So, the equation becomes:

$$2H(u,v) = T(u,v) + T(8-u,v) + T(u,8-v) - T(8-u,8-v) \tag{4.27}$$

Hence 2-D DHT of an 8x8 matrix can be computed using 8-point 1-D DHT.

4.5.2 Architecture

The figure 4.5 illustrates the design flow system to implement the architecture of the 2-D DHT. The architecture has been implemented using the 1-D DHT blocks as components and various shift registers to smoothly run the entire operation. The input matrix has to be fed row-wise to the FPGA since it cannot take such a large input matrix at a time.

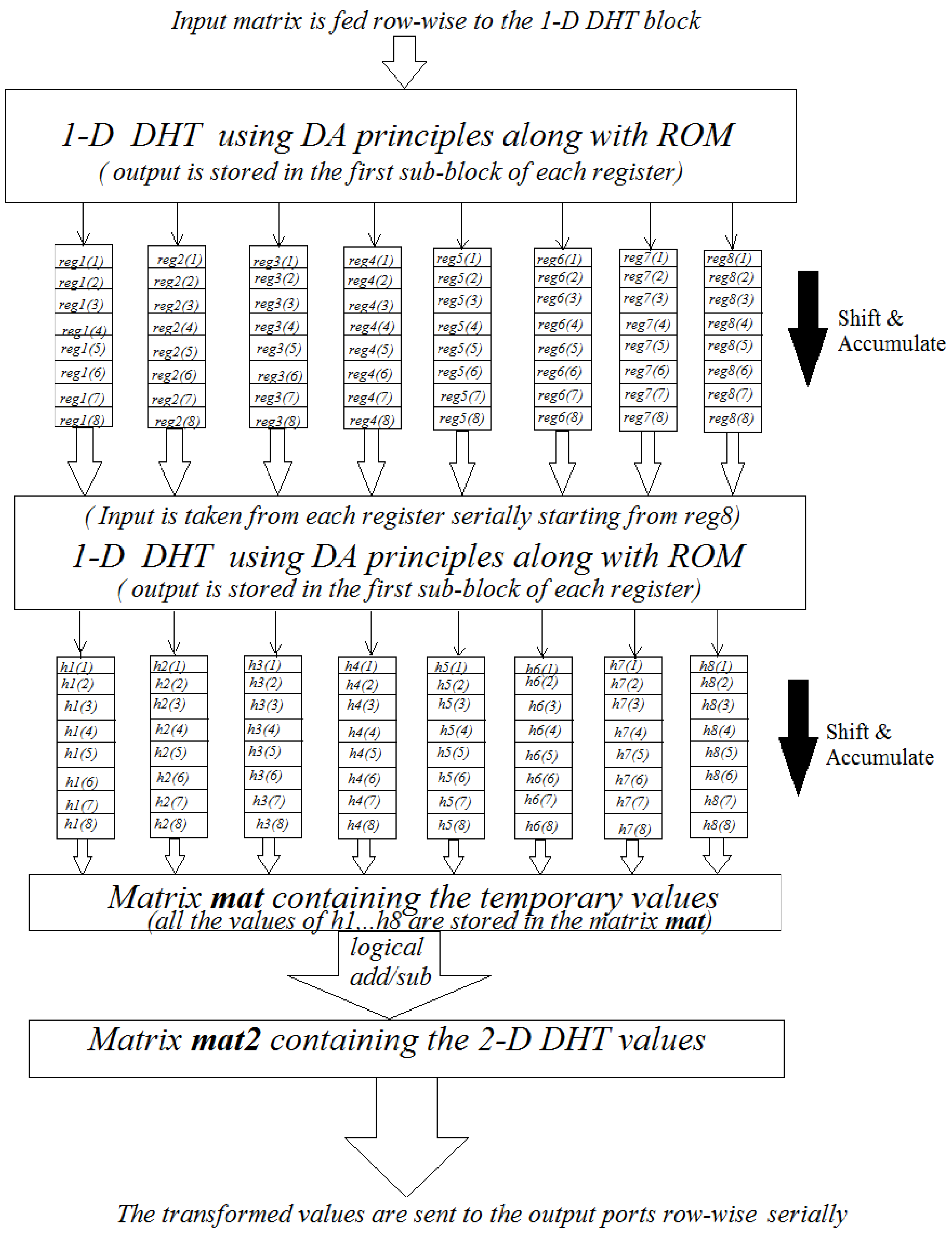

Input matrix is fed row-wise to the 1-D DHT block
1-D DHT using DA principles along with ROM
(output is stored in the first sub-block of each register)
reg1(1) … reg8(8)
Shift & Accumulate
(Input is taken from each register serially starting from reg8)
1-D DHT using DA principles along with ROM
(output is stored in the first sub-block of each register)
h1(1) … h8(8)
Shift & Accumulate
Matrix mat containing the temporary values
(all the values of h1...h8 are stored in the matrix mat)
logical add/sub
Matrix mat2 containing the 2-D DHT values
The transformed values are sent to the output ports row-wise serially

Figure 4.5: Flow diagram of 2-D DHT implemntation

4.5.3 Working

Implementation of 2-D DHT is done using state machines. The 8x8 matrix input is fed to the 1-D DHT block row-wise after a certain delay for the computation of transform coefficients of each row. The transformed coefficients are stored in various registers arrays and they are shifted after each row transform is computed, so that finally all the transformed values can be stored and located for further computation, i.e. it works on shift and accumulate method.

After the first DHT is applied to the rows; consisting of eight number of 8-bit input vectors; they are transformed and are stored in eight registers of 10-bit vectors. Similarly the columns are fed for DHT computation to obtain the temporary outcome. Another block is used for DHT computation this time, which computes DHT of 10-bit vectors to give 12-bit vectors. The transformed values are again shifted and stored in the array of 12-bit registers to obtain the temporary matrix **T**.

Finally, the temporary outcome are added and subtracted according to the logic given in eq. (4.27) to obtain the desired output matrix **mat** i.e.8x8 2-D DHT of the input 8x8 matrix. All the steps are executed in different states like computation of DHT of the rows/columns of input vectors, shifting of the transformed values in the register arrays and calculating the DHT from temporary values in registers.

Chapter 5

RESULTS AND DISCUSSION

5.1 MATLAB Simulation Results

Matlab code was written for image compression using energy quantization technique explained in section 2.3. The images were reconstructed and the performance parameters such as mean square error (MSE) and peak signal to noise ratio (PSNR) were calculated. Source coding was not implemented and the images were reconstructed from the quantized values only. The code was then tested on two bit-map image files and the results are tabulated below.

Table 5.1: MSE and PSNR tabulated for Lena and Baboon images.

IMAGE	**Threshold value as % of normalized energy**	**10%**	**20%**	**40%**	**60%**	**80%**	**100%**
LENA	MSE	21.4292	31.7735	45.7513	55.7382	66.5523	69.6302
	PSNR	80.1777	76.2389	72.5931	70.6186	68.8454	68.3933
BABOON	MSE	67.5956	101.201	140.805	165.766	184.946	204.709
	PSNR	68.6898	64.6541	61.3515	59.7195	58.6246	57.6094

Original images are of size 512x512 pixels:

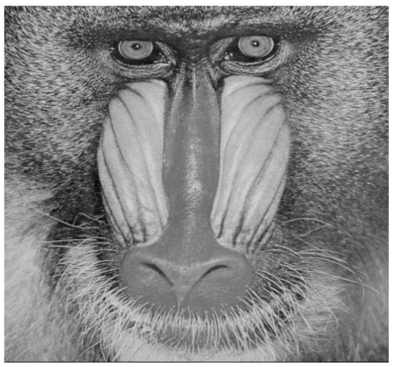
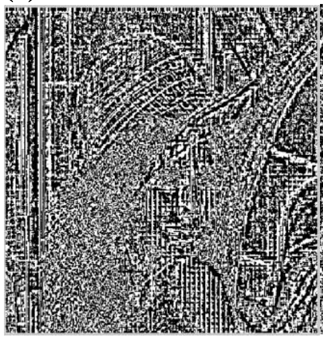
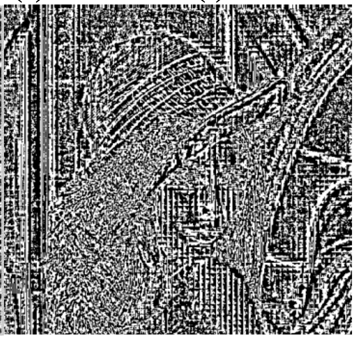
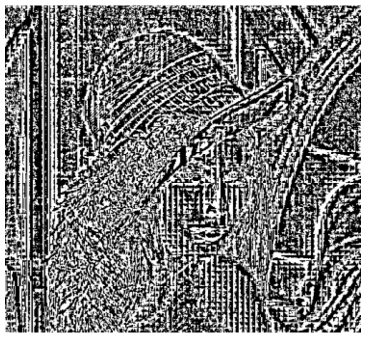


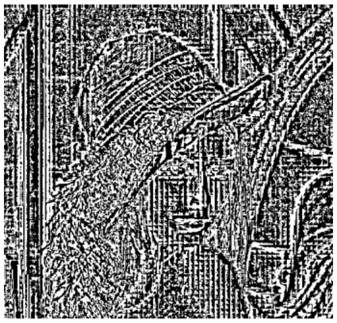
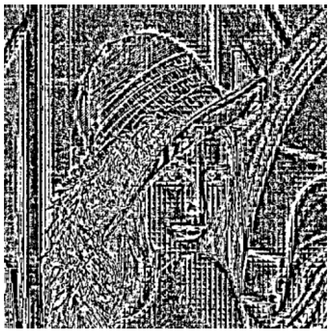
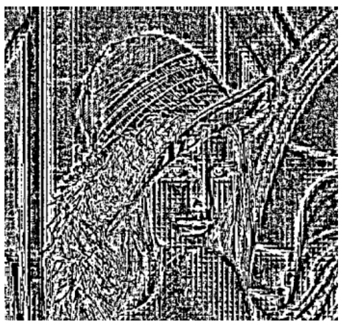
Figure 5.1:Lena original bitmap imageFigure 5.2: Baboon original bitmap image

(a) Eth=10%En (b) Eth=20%En (c) Eth=40%En

(d)Eth=60%En (e) Eth=80%En(f) Eth=100%En

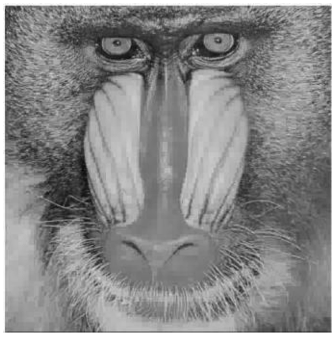
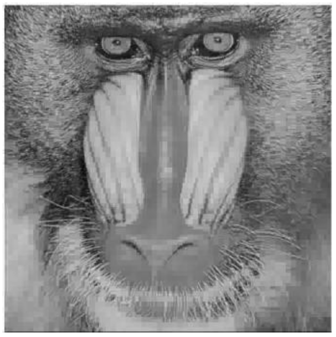
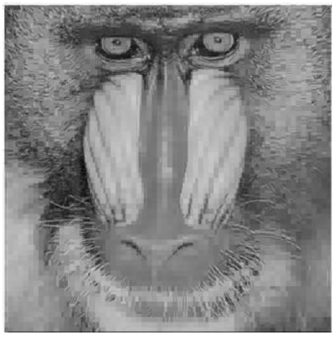




(g)Eth=10%En(h) Eth=20%En(i) Eth=40%En

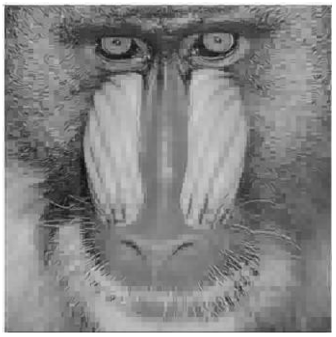
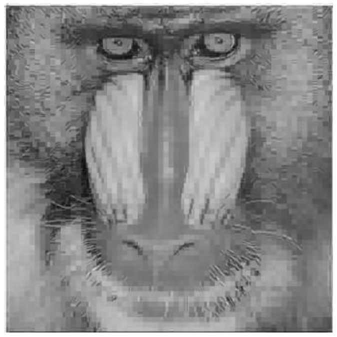
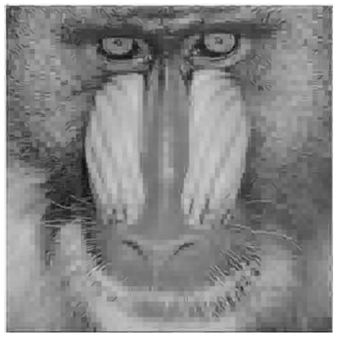




(j)Eth=60%En (k) Eth=80%En(l) Eth=100%En

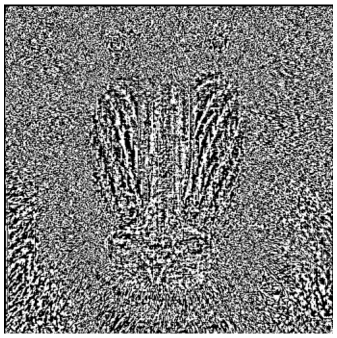
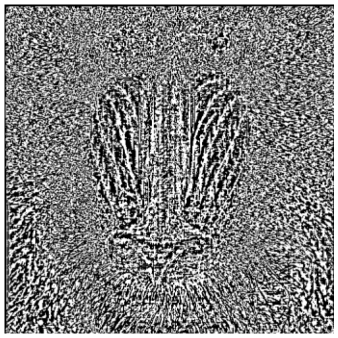
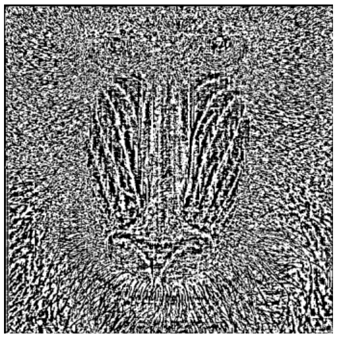
Figure 5.3: Reconstructed Lena Images and the error images with the threshold values given as a percentage of normalized energy of the image.(a-f: reconstructed images; g-l: error images)

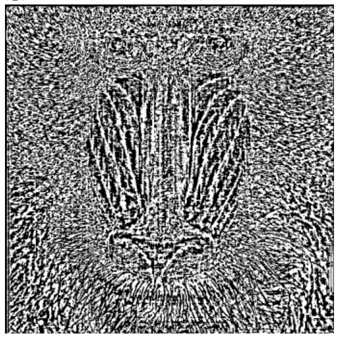
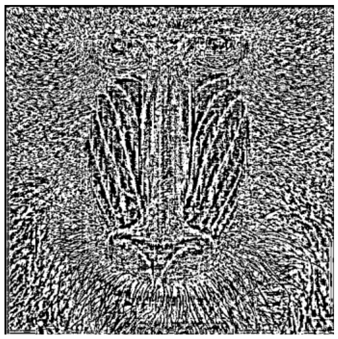
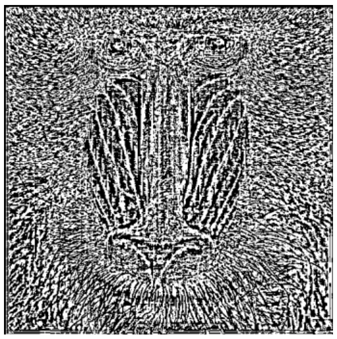
(a)Eth=10%En(b) Eth=20%En(c)Eth=40%En

(d)Eth=60%En (e)Eth=80%En(f)Eth=100%En

(g)Eth=10%En(h)Eth=20%En(i)Eth=40%En

(j)Eth=60%En (k)Eth=80%En(l)Eth=100%En

Figure 5.4: Reconstructed Baboon Images and the error images with the threshold values given as percentage of normalized energy of the image(a-f: reconstructed images; g-l: error images)

5.2 Xilinx Simulation Results and Discussion

5.2.1 Design Summary for different Architectures

1. Systolic Architecture for 3-point DHT

Table 5.2: Design Summary for SA architecture for 3-point DHT

Logic utilization	Used	Available	Utilization
Number of slices	428	4656	9%
Number of slice flip-flops	359	9312	3%
Number of 4-input LUTs	783	9312	8%
Number of bonded IOBs	73	232	31%
Number of GCLKs	1	24	4%

2. DistributedArithmeticArchitecture for 3-point DHT

Table 5.3: Design Summary for DA architecture for 3-point DHT

Logic utilization	Used	Available	Utilization
Number of slices	121	4656	2%
Number of slice flip-flops	143	9312	1%
Number of 4-input LUTs	208	9312	2%
Number of bonded IOBs	77	232	33%
Number of GCLKs	1	24	4%

We can clearly see that hardware used for DA architecture is much less than SA architecture. This difference will only increase as we increase the number of inputs i.e. for 8-point DHT the silicon used in DA will be much less than that used in SA. Also the power consumption is much lesser in DA than in SA architecture. This is due to the reason that there is no multiplication or any other higher calculations involved in DA model. It comprises only of adders and shift registers. Hence DA architectures are faster, low power and more compact than SA architectures.

3. 8- point DHT using two 4-point modules in pipelined stages

Table 5.4: Design Summary for architecture of 8-pt DHT using two 4-pt modules

Logic utilization	Used	Available	Utilization
Number of slices	516	13697	3%
Number of slice flip-flops	666	27392	2%
Number of 4-input LUTs	795	27392	2%
Number of bonded IOBs	195	556	35%
Number of GCLKs	1	16	6%

4. 8- point DHT usingDA principleswithROM (Input is 8 bit vector and Output is 10 bit vector)

Table 5.5: Design Summary for architecture of 8-pt DHT by DA (8-bit to 10-bit)

Logic utilization	Used	Available	Utilization
Number of slices	349	13697	2%
Number of slice flip-flops	307	27392	1%
Number of 4-input LUTs	600	27392	2%
Number of bonded IOBs	147	556	26%
Number of GCLKs	1	16	6%

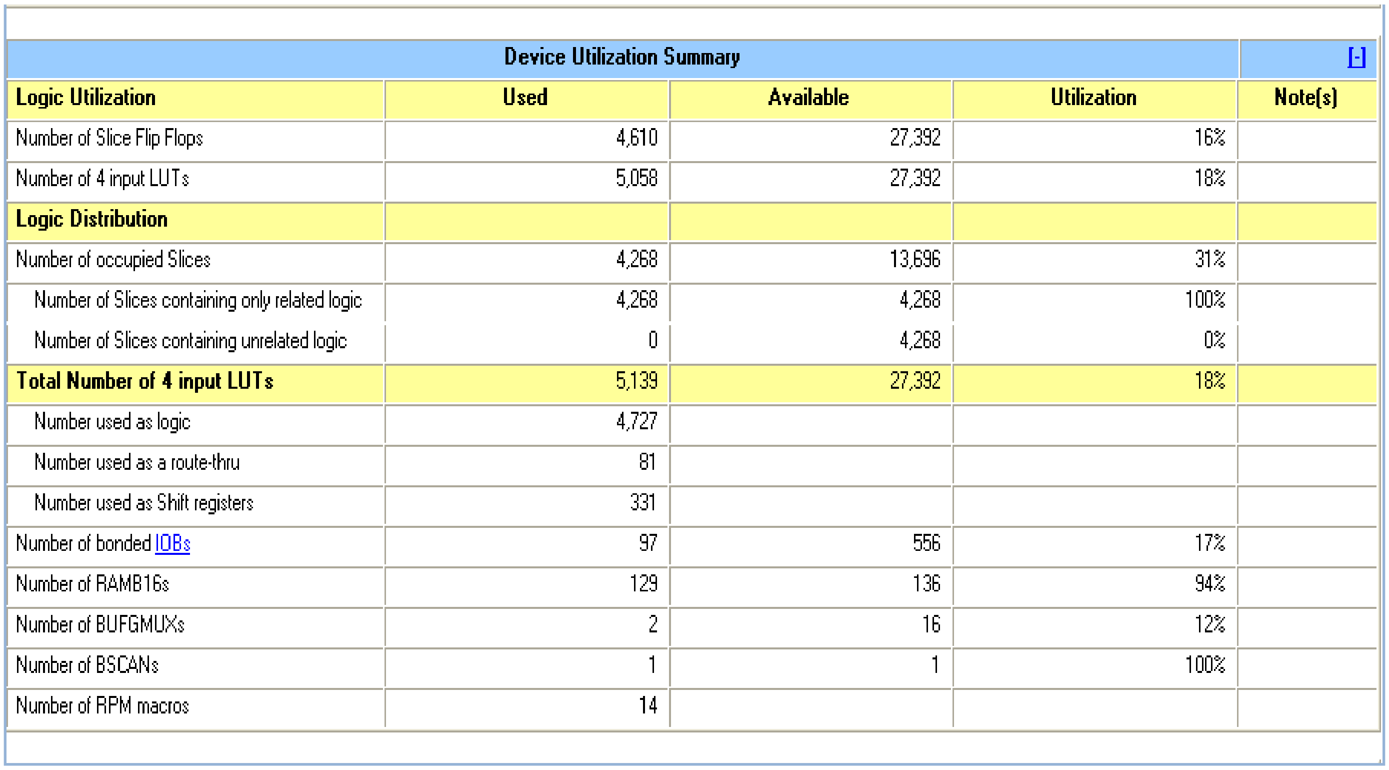
5. 8- point DHT using DA principleswithROM (Input is 10 bit vector and Output is 12 bit vector)

Table 5.6: Design Summary for architecture of 8-pt DHT by DA (8-bit to 10-bit)

Logic utilization	Used	Available	Utilization
Number of slices	432	13697	3%
Number of slice flip-flops	355	27392	1%
Number of 4-input LUTs	694	27392	2%
Number of bonded IOBs	179	556	32%
Number of GCLKs	1	16	6%

The silicon utilization in the architecture with pipelined stages is higher than in the architecture using only DA principles with ROM. Also the computational time is more in pipelined stage architecture due to the delays introduced in the model of the method. Hence, architecture based on DA principles using ROM is more efficient and is used for the modeling of 2-D DHT. Another architecture which inputs 10-bits vectors and gives 12-bit vectors is also implemented, which also is more efficient than the pipelined stage architecture.

6. 2-D DHT OF 8X8 INPUT MATRIX



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	4,610	27,392	16%	
Number of 4 input LUTs	5,058	27,392	18%	
Logic Distribution				
Number of occupied Slices	4,268	13,696	31%	
Number of Slices containing only related logic	4,268	4,268	100%	
Number of Slices containing unrelated logic	0	4,268	0%	
Total Number of 4 input LUTs	5,139	27,392	18%	
Number used as logic	4,727			
Number used as a route-thru	81			
Number used as Shift registers	331			
Number of bonded IOBs	97	556	17%	
Number of RAMB16s	129	136	94%	
Number of BUFGMUXs	2	16	12%	
Number of BSCANs	1	1	100%	
Number of RPM macros	14			

Figure 5.5: Design Summary of the 2-D DHT design using VHDL as the synthesis tool

5.2.2Power Analysis

The total estimated power consumption of the 2-D DHT architecture design is 103 mW.

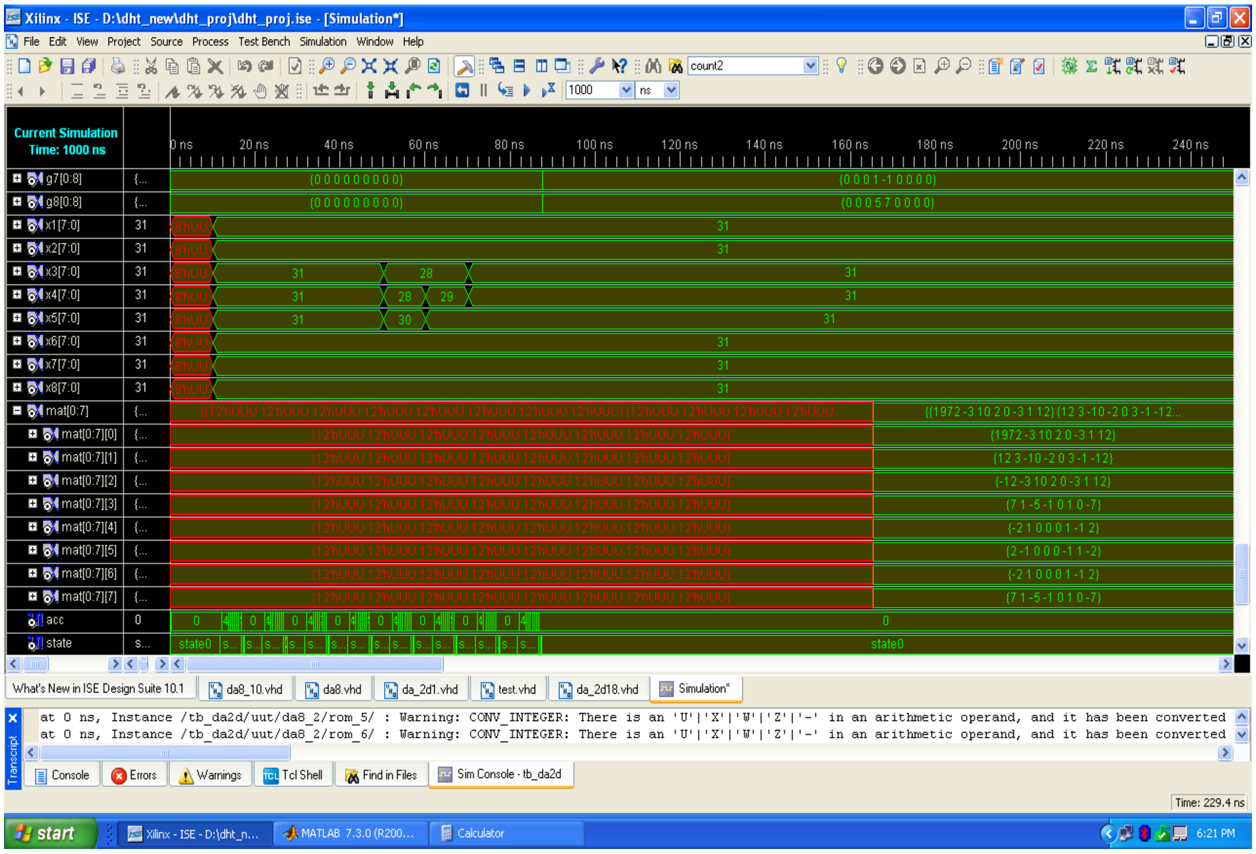
Table 5.7: Power analysis of the 2-D architecture

Power summary	**I(mA)**	**P(mW)**
Total estimated power consumption	--	103
Total Vccaux 2.50V	10	25
Total Vcco25 2.50V	1	3
Quiescent Vccint 1.50V	50	75
Quiescent Vccaux 2.50V	10	25
Quiescent Vcco25 2.50V	1	3

5.2.3Comparison between the Matlab and VHDL outputs obtained for 2-D DHT.

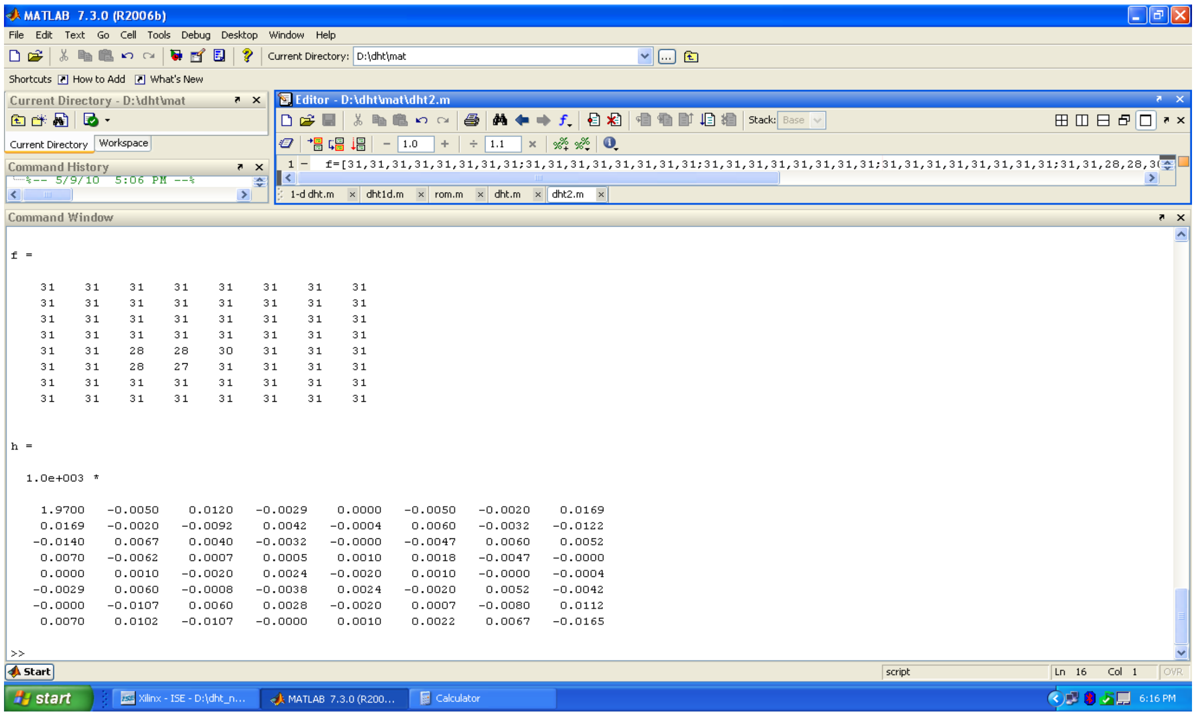
2-D DHT is calculated for 8x8 matrices inMatlab and Xilinx. The simulation results of both are shown for two different inputs. The matrix **mat** contains the 2-D DHT for the input given to **x1-x8** registers in the Xilinx simulation results shown below. For Matlab simulation, **f** contains the input matrix and **h** gives the output matrix.

Figure 5.6: Simulation result for the 1st input matrix


Xilinx - ISE - D:\dht_new\dht_proj\dht_proj.ise - [Simulation*]
File Edit View Project Source Process Test Bench Simulation Window Help
Current Simulation Time: 1000 ns
0 ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns 220 ns 240 ns
What's New in ISE Design Suite 10.1
da8_10.vhd
da8.vhd
da_2d1.vhd
test.vhd
da_2d18.vhd
Simulation*
Console
Errors
Warnings
Tcl Shell
Find in Files
Sim Console - tb_da2d
Time: 229.4 ns
start
Xilinx - ISE - D:\dht_n...
MATLAB 7.3.0 (R200...
Calculator
6:21 PM

```
g7[0:8]        {0 0 0 0 0 0 0 0 0}   {0 0 0 1 -1 0 0 0 0}
g8[0:8]        {0 0 0 0 0 0 0 0 0}   {0 0 0 5 7 0 0 0 0}
x1[7:0]   31   8'hUU  31
x2[7:0]   31   8'hUU  31
x3[7:0]   31   8'hUU  31  28  31
x4[7:0]   31   8'hUU  31  28  29  31
x5[7:0]   31   8'hUU  31  30  31
x6[7:0]   31   8'hUU  31
x7[7:0]   31   8'hUU  31
x8[7:0]   31   8'hUU  31
mat[0:7]       {{1972 -3 10 2 0 -3 1 12}{12 3 -10 -2 0 3 -1 -12...
mat[0:7][0]    {1972 -3 10 2 0 -3 1 12}
mat[0:7][1]    {12 3 -10 -2 0 3 -1 -12}
mat[0:7][2]    {-12 -3 10 2 0 -3 1 12}
mat[0:7][3]    {7 1 -5 -1 0 1 0 -7}
mat[0:7][4]    {-2 1 0 0 0 1 -1 2}
mat[0:7][5]    {2 -1 0 0 0 -1 1 -2}
mat[0:7][6]    {-2 1 0 0 0 1 -1 2}
mat[0:7][7]    {7 1 -5 -1 0 1 0 -7}
acc       0    0
state     s... state0
```

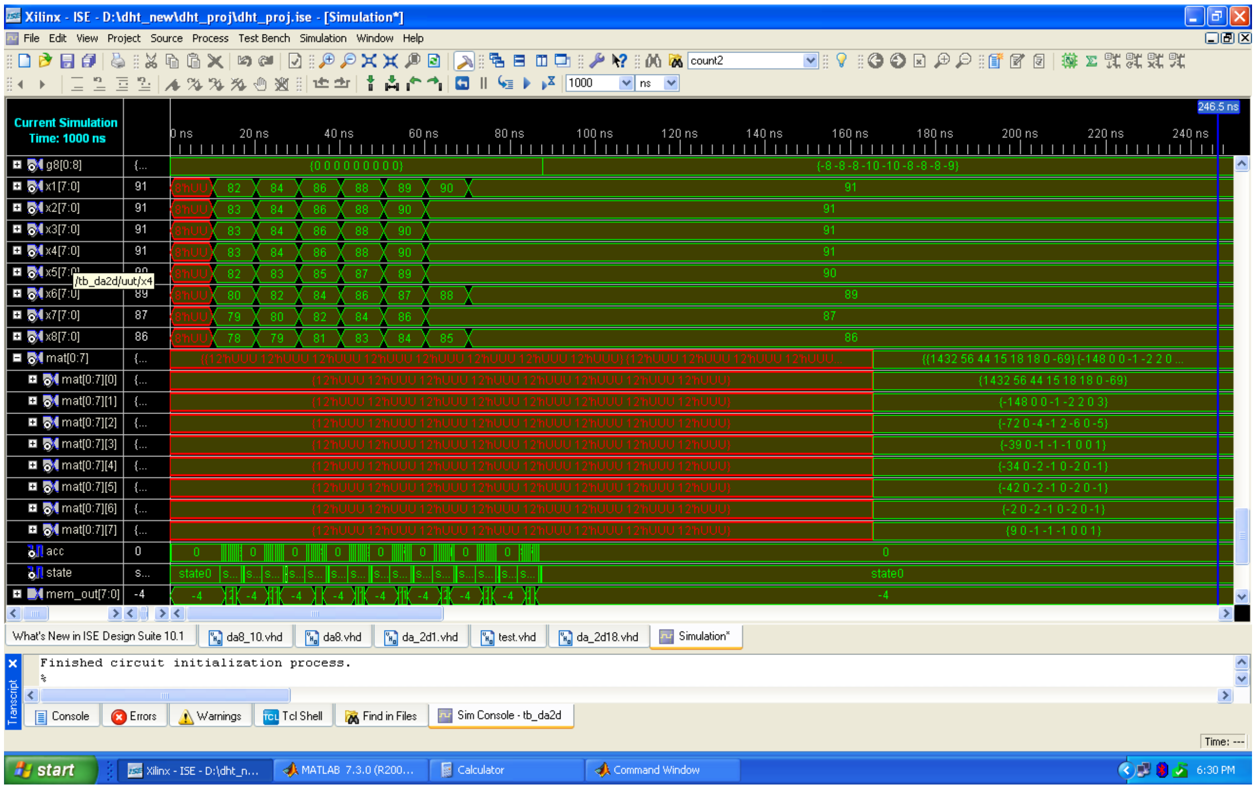
```
at 0 ns, Instance /tb_da2d/uut/da8_2/rom_5/ : Warning: CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has been converted
at 0 ns, Instance /tb_da2d/uut/da8_2/rom_6/ : Warning: CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has been converted
```



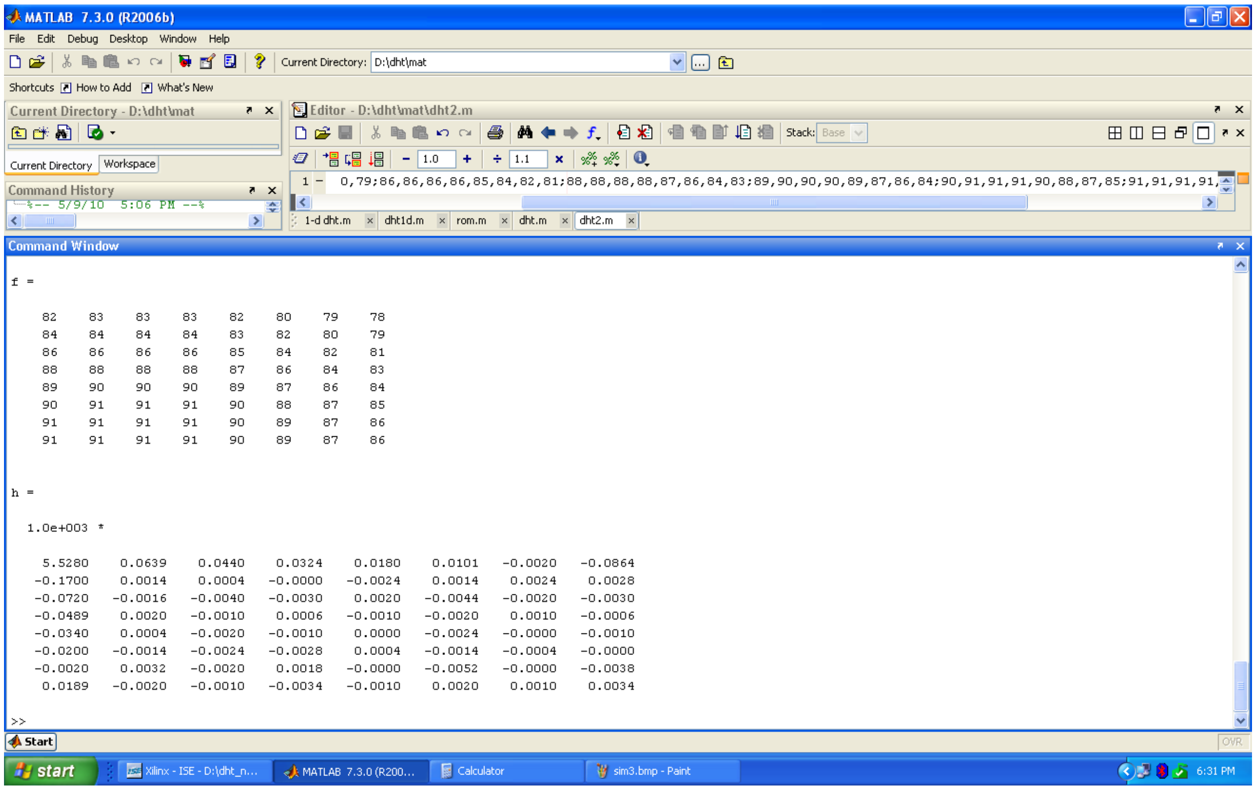
```
f=[31,31,31,31,31,31,31,31;31,31,31,31,31,31,31,31;31,31,31,31,31,31,31,31;31,31,31,31,31,31,31,31;31,31,28,28,3
```

```
f =

    31    31    31    31    31    31    31    31
    31    31    31    31    31    31    31    31
    31    31    31    31    31    31    31    31
    31    31    31    31    31    31    31    31
    31    31    28    28    30    31    31    31
    31    31    28    27    31    31    31    31
    31    31    31    31    31    31    31    31
    31    31    31    31    31    31    31    31


h =

  1.0e+003 *

    1.9700   -0.0050    0.0120   -0.0029    0.0000   -0.0050   -0.0020    0.0169
    0.0169   -0.0020   -0.0092    0.0042   -0.0004    0.0060   -0.0032   -0.0122
   -0.0140    0.0067    0.0040   -0.0032   -0.0000   -0.0047    0.0060    0.0052
    0.0070   -0.0062    0.0007    0.0005    0.0010    0.0018   -0.0047   -0.0000
    0.0000    0.0010   -0.0020    0.0024   -0.0020    0.0010   -0.0000   -0.0004
   -0.0029    0.0060   -0.0008   -0.0038    0.0024   -0.0020    0.0052   -0.0042
   -0.0000   -0.0107    0.0060    0.0028   -0.0020    0.0007   -0.0080    0.0112
    0.0070    0.0102   -0.0107   -0.0000    0.0010    0.0022    0.0067   -0.0165

>>
```

Figure 5.7: Simulation result for the 2nd input matrix


Xilinx - ISE - D:\dht_new\dht_proj\dht_proj.ise - [Simulation*]
Current Simulation Time: 1000 ns
g8[0:8]
x1[7:0]
x2[7:0]
x3[7:0]
x4[7:0]
x5[7:0]
x6[7:0]
x7[7:0]
x8[7:0]
mat[0:7]
acc
state
mem_out[7:0]

```
Finished circuit initialization process.
%
```



```
1 -   0,79;86,86,86,86,85,84,82,81;88,88,88,88,87,86,84,83;89,90,90,90,89,87,86,84;90,91,91,91,90,88,87,85;91,91,91,91,
```

```
f =

    82    83    83    83    82    80    79    78
    84    84    84    84    83    82    80    79
    86    86    86    86    85    84    82    81
    88    88    88    88    87    86    84    83
    89    90    90    90    89    87    86    84
    90    91    91    91    90    88    87    85
    91    91    91    91    90    89    87    86
    91    91    91    91    90    89    87    86


h =

  1.0e+003 *

    5.5280    0.0639    0.0440    0.0324    0.0180    0.0101   -0.0020   -0.0864
   -0.1700    0.0014    0.0004   -0.0000   -0.0024    0.0014    0.0024    0.0028
   -0.0720   -0.0016   -0.0040   -0.0030    0.0020   -0.0044   -0.0020   -0.0030
   -0.0489    0.0020   -0.0010    0.0006   -0.0010   -0.0020    0.0010   -0.0006
   -0.0340    0.0004   -0.0020   -0.0010    0.0000   -0.0024   -0.0000   -0.0010
   -0.0200   -0.0014   -0.0024   -0.0028    0.0004   -0.0014   -0.0004   -0.0000
   -0.0020    0.0032   -0.0020    0.0018   -0.0000   -0.0052   -0.0000   -0.0038
    0.0189   -0.0020   -0.0010   -0.0034   -0.0010    0.0020    0.0010    0.0034

>>
```

We can see that the results obtained from the hardware implementation are same for the odd numbered columns and a little error is generated in even-numbered columns. This is due to the reason that the even-numbered columns of the kernel matrix generated from the ‘cas’ function has mixed fraction values which have been approximated. This approximation can be reduced if we use binary representation of decimal values for calculation purposes.

Also if the input matrix contains large values then, the transformed coefficients overflow the registers and as a result an error is generated. It can be seen in the second simulation that the first value of the transformed matrix has quite deviated from the required value. It can be corrected if registers of larger number of bits are used to store the transformed values.

Chapter 6

CONCLUSION AND FUTURE WORK

In the present work, two-dimensional Discrete Hartley Transform for an 8x8 input matrix was implemented in FPGA using VHDL as the synthesis tool. The 1-D DHT was also calculated for 8-point input using two algorithms and their effectiveness were discussed. It is shown that the DA approach provides better performance in terms of speed and area when is compared with the pipelined approach. This primarily focuses on image compression with less computation and low power. The simulation results and design summary for 2-D DHT were obtained andit was shown that the architecture implemented is an efficient method which uses limited space and time. The hardware utilization is quite optimum and power analysis shows that the power requirement is also optimum.

However if the input contents are large, they tend to overflow from the registers and hence error occurs. It can be rectified by saving the transformed coefficients in larger registers. Also due to quantization in the contents of the ROM, even-number outputs are more deviated from the desired results than the odd-numbered outputs. This is due to the reason that even numbered columns of the transform kernel consist of mixed fractions which are rounded off to be store in the ROM registers. This drawback can be removed if the decimal fractions are converted to binary representation before being stored. Also, a lot of memory is used in this architecture. It can be solved by using the ROM-free DA technique. These are some of the improvements that can be done to the improvise the design.

REFERENCES:

[1] S.K.Pattanaik and K.K.Mahapatra,"DHT Based JPEG Image Compression Using a Novel Energy Quantization Method"*IEEE International conference on industrial technology,* pp. 2827 – 2832, Dec 2006 .

[2] R.C. Gonzalez, R.E. Woods, Digital Image Processing, Pearson Education 3rd Edition 2008.

[3] F. Vahid and T. Givargis, Embedded system design: A unified hardware/software introduction, Wiley India (P.) Ltd, 3rd edition 2009.

[4] A. Amira, "An FPGA based system for discrete hartley transforms*." IEEE publication,* pp. 137-140, 2003

[5] P.K.Meher, S. Thambipillai and J.C. Patra, "Scalable and modular memory-based systolic architectures for discrete hartley transform"*IEEE Transactions on cirucits and systems-I:regular papers, Vol53,* pp. 1065-1077, May 2006

[6] RN.Bracewell, 0.Buneman, H. Hao and **J.** Villasenor, "Fast two-dimensional hartley transform*" Proceedings of IEEE ,* Vol 74, No. 9, Sept1986

[7] CR. Baugh and BA. Wooley, "A two's complement parallel aaray multiplication algorithm"*IEEE Transactions on computers,* Vol C-22,pp. 1045-1047, Dec 1973

[8] Bracewell, Ronald N. "The Hartley transform" *New York: Oxford university press* 1986

[9] Ranjan Bose, Information theory coding and Cryptography, Tata McGraw-Hill 2003.

[10]C. H. Paik and M. D. Fox, "Fast Hartley transform for image processing,"*IEEE Trans. Med. Image, vol. 7,* no. 6, pp. 149–153, Jun. 1988.

[11] H.S. Hou, "The fast Hartley transform algorithm," *IEEE Transactions on Computers,* vol. C-36, no. 2, pp. 147–156, Feb. 1987.

[12] L.W. Chang and S.W. Lee, "Systolic arrays for the discrete Hartleytransform," *IEEE Transactions Signal Processing,* vol. 39, no. 11, pp. 2411–2418,Nov. 1991.

[13] P. K. Meher and T. Srikanthan, “A scalable and multiplier-lessfully-pipelined architecture for VLSI implementation of discreteHartley transform,” in *Proc. Int. Symp. Signals, Circuits Syst. (SCS’03),* vol. 2, Jul. 10–11, 2003, pp. 393–396.

[14] N. Kihara, et al., "The Electronic Still Camera: A New Concept inPhotography," *IEEE Trans. Cons. Electron.*, Vol. CE-28, NO. 3, pp. 325-335, Aug. 1982.