SPIHT IMAGE COMPRESSION AND RECONSTRUCTION ON FIELD PROGRAMMABLE GATE ARRAY

Submitted by

Ursila Tanweer Khan

Supervisor:

Dr. Arshad Aziz

Thesis

Submitted to the Department of Electronic and Power Engineering

College of Marine Engineering (PNEC), Karachi

National University of Sciences and Technology, H-12, Islamabad

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

With Specialization in Communication

June 2011

بسم الله الرحمن الرحيم

IN THE NAME OF ALLAH SUBHANAHU WA TA’ALA, THE MOST GRACIOUS, MOST MERCIFUL

ABSTRACT

This thesis presents the implementation of the Set Partitioning in Hierarchical Trees (SPIHT) Image Compression Algorithm on Reconfigurable platform. For flexibility of design and to achieve optimized results, we have combined the high-level utility of MATLAB with the flexibility and optimization of FPGA to implement this wavelet-based image compression coder on a sample image. The design is a pipe-lined architecture comprising of four phases in which Phase 1 and Phase 4 have been implemented on software and Phase 2 and Phase 3 on hardware. The output of each phase serves as the input to the succeeding phase. Our hardware architecture has been coded using Verilog HDL and our target devices are from Spartan 3E and Virtex-4 Family developed by Xilinx.

In Phase 1, we have computed the 4-level DWT of the image using Daubechies wavelets which serve as input to the next phase. In Phase 2, we have designed the SPIHT encoder to code the wavelets. In Phase 3, the decoding of the wavelets takes place and finally in Phase 4, IDWT is carried out to reconstruct the image. The SPIHT encoder and decoder are designed on hardware and contain four modules. Our hardware modules are implemented on cost effective devices and are giving an appreciable threshold while occupying area resources efficiently.

I dedicate my work to my parents Tanweer Ahmad Khan and Nuzhat Tanweer without whom I would not be where I stand today.

ACKNOWLEDGEMENTS

I begin with thanking The Almighty Allah (Subhanahu-Wa-Ta'ala) for granting me His Never-ending Blessings and His extreme Kindness all of which contribute to where I stand today. It was always a simple prayer to the Almighty that made the impossible appear simple and possible.

I want to express my gratitude to my parents and my brothers who never lost faith in me and encouraged me during the most difficult times of my work. It is their support and patience because of which I passed major roadblocks.

I am heartily thankful to my advisor Dr. Arshad Aziz who guided and motivated me throughout my research work and kept me going even when the situation appeared very bleak. It's his continuous driving force that has made the completion of my thesis possible.

During my daily work, I was blessed with people who were always supportive in some way or the other. I express my sincere gratitude to Dr. Fawad Ahmed who always helped me during the times when I faced serious problems in my work.

I thank all the Guidance Committee members Dr. Pervez Akhtar, Dr. Valiuddin Abbas and Dr. Bilal Kadri. I am also thankful to Cmdr(R) Riaz Mahmud, for always offering me warm greetings, words of encouragement and most importantly his seat, for endless hours during my work.

Lastly, I offer my regards to all of those including my classmates who supported me in any way during the completion of my research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Study Background

Image Compression Algorithms have been the answer to efficient and easy storage and transmission of image data for a long time. Various methods of Compression Algorithms have been the focus of research among which wavelet-based compression algorithms are found to have given better performance and highest compression ratios compared to other types like Vector Quantization, Fractals and DCT. Some popular image compression algorithms are;

- EZW (Embedded zero-tree wavelet algorithm)
- SPIHT (Set Partitioning in Hierarchical Trees)
- JPEG (Joint Photographic Experts Group)

Most Image Compression algorithms have high memory requirement and frequent modifications and alterations are required when designing the architecture. For this reason, a Reconfigurable Hardware i.e. The FPGA (Field-Programmable Gate Array) is a suitable option because it offers immense flexibility while maintaining speed, utilizing resources and optimizing performance.

In this work, the SPIHT (Set Partitioning in Hierarchical Trees) Image Compression technique which was developed by Amir Said and William Pearlman has been explored and implemented on an FPGA [1]. We have combined software and hardware platforms to implement this algorithm and achieved good results.

1.2 Thesis Scope

In this thesis work, efforts are made to implement the SPIHT algorithm using both software and hardware platforms to achieve maximum optimization. The SPIHT can inherently be split into two phases. The first phase of calculating the wavelet coefficients have been implemented using software high-level language MATLAB. For the computationally intensive part of encoding and decoding, we employed the Hardware Description Language Verilog HDL for FPGA implementation. Our target platforms are the XC3S100E from Spartan 3E family and XC4VLX80 from Virtex- 4 family, both of which are developed by Xilinx®.

1.3 Thesis Structure

Chapter 2 briefly covers introduction to data compression, image compression and its area of application.

In Chapter 3, we have discussed the Discrete Wavelet Transform and how it enhances the performance of compression algorithms.

In Chapter 4, we have discussed briefly, the background of our Reconfigurable Hardware Platform i.e. The Field-Programmable Gate Array and the internal architecture of the Xilinx FPGA.

Chapter 5 gives a detailed description of the basic SPIHT algorithm along with its characteristics and basic working mechanism.

In Chapter 6 we have discussed in detail, our design architecture and the integration of software and hardware to optimize the performance of the SPIHT algorithm. This chapter also summarizes the implementation results of FPGA modules.

In Chapter 7, we have summed up our implementation results and its comparison with the other known SPIHT implementations.

Finally, in Chapter 8, conclusions, future possibilities and improvements of this work have been discussed and how this work can be extended to achieve further optimization.

CHAPTER 2

INTRODUCTION TO DATA COMPRESSION

2.1 Introduction

Over the last decade, the growing need of data compression has been felt in every aspect of our lives. As technology has progressed, researchers have adopted newer and better forms of compression schemes and today there are very few areas where we do not see the obvious prevalence of data compression.

Uncompressed multimedia data such as graphics, audio and video require substantial storage capacity and transmission bandwidth. Although the performance of modern systems, processor speeds and mass-storage capacity has seen dramatic increase, there is always room for higher bandwidth and storage capacity [2]. Compression has proved to be a solution to all the above mentioned problems and has therefore become an integral part of our daily lives.

All around us, we find different forms of data compression in a diverse range of applications. All the images that we find on the Internet are compressed mostly in the JPEG or GIF format, while the PNG format is also observed in certain places. High-Definition TV is another example in which image data is compressed using MPEG-2 format. Internet browsers facilitate and optimize browsing in terms of time by making use of compression in its progressive image transmission when opening a web page.

Data compression involves primary reliance on the fact that the data is redundant and that by removing its redundancy, it can be transformed to minimize the size of its representation. [3] It reduces the size of the data blocks to be transmitted over a network link enabling increase in media channel capacity.

The mechanism of data compression incorporates a coding scheme at each end of a transmission link. At the sender end of the link, the coding scheme removes certain redundant data from the data blocks and accordingly replaces them correctly at the receiver end. Redundancy removal at the sender side reduces, both the size of data blocks and bandwidth requirement, thus allowing greater volumes of data to be transmitted at a time. A compressor reduces the size of a file by deciding which data is more frequent and assigns it less bits than to less frequent data. Amount of compression that a certain algorithm offers varies depending upon the type of media (audio, video, images or text), size of the file and type of compression. A typical compression scheme is shown in Figure 2.1.

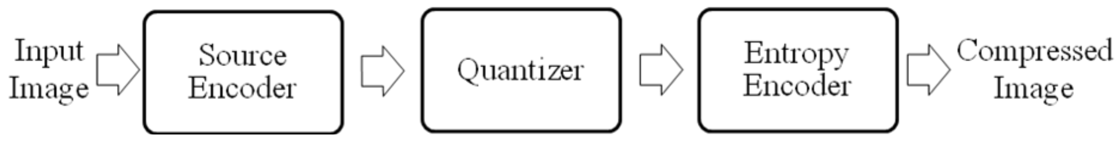


Figure 2.1 Typical Compression Scheme

2.2 Image Compression Schemes

Image compression is different than compressing binary data in that its purpose is to remove the redundant data from an image in order to make it area and transmission time efficient. If general purpose compression programs are used to compress images, the result can be optimized only to a certain degree. Encoders which are designed for such compression techniques exploit the statistical properties of images. Image compression may be lossy or lossless. In case of lossy compression the finer details of an image can be compromised for saving bandwidth or storage capacity. However, in case of lossless compression, perfect reconstruction of all image data is required at the receivers end with no loss of fine details and the compressed data on recovery must be an exact replica of the original data. This is the case in compression of binary data such as executables, documents etc which need to be exactly reproduced after decompression. [4]

Lossy compression may be applied to multimedia data such as images and music which does not require exact replication as long as there is tolerable error between the original signal and the compressed form and insignificant or no audible or visual loss is present. In case of an image file, an approximation of the original image can be used for most purposes except medical imaging which requires absolute recovery of all information.

2.3 Applications

Media can be in form of text, audio, video and still images. Audio, Video and Image compression schemes are designed to reduce the transmission bandwidth requirement of digital streams and the storage size of files. [5] Data compression finds direct applications in MP3 players and computers. Digitally compressed audio streams are applied in most video DVDs; digital television, media streaming on the internet and in terrestrial radio broadcasts. Compression is used in digital cameras, to increase storage capacities with tolerable degradation of picture quality. Similarly, DVDs use MPEG-2 Video codec for video compression.

CHAPTER 3

DISCRETE WAVELET TRANSFORM

3.1 Introduction

Wavelet Transform has gained widespread recognition and acceptance in the fields of Image compression and signal processing. The DWT (Discrete Wavelet Transform), is multi-resolution in nature and on account of this property, it has benefits in application areas where scalability and tolerable degradation are important.

3.2 DWT Basics

The wavelet transform decomposes a signal into a set of basic function called wavelets. A discrete wavelet transform (DWT) is a wavelet transform for which the wavelets are sampled at predefined units of time. There are several types of wavelets among which some of the commonly used ones are discussed in the following sections.

3.2.1 Haar Wavelets

The first DWT was invented by the Hungarian mathematician Alfréd Haar. The Haar wavelets transform works on an input represented by a list of 2^n numbers. It pairs up the input values, retains the difference and passes the sum. This process is repeated recursively, pairing up the sums to provide the next scale: finally resulting in $2^n - 1$ differences and one final sum.

The Haar wavelet's mother wavelet function $\psi(t)$ can be described as

$$\psi(t) = \left\{ \begin{array}{cc} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & otherwise \end{array} \right\}$$

Its scaling function $\varphi(t)$ can be described as;

$$\varphi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & otherwise \end{cases}$$

3.2.2 Daubechies Wavelets

Daubechies wavelets were formulated by the Belgian mathematician Ingrid Daubechies in 1988 and are the most widely used set of discrete wavelet transforms. Many variations of the original wavelets have now been developed and are largely used in image compression schemes. The Daubechies wavelets generate finer discrete samples of a mother wavelet functions progressively. The wavelets make use of the recurrence relations between the scaling and wavelet functions such that each resolution is twice that of the previous scale. [7] We can take the example of Daubechies D4 wavelets which have four scaling function coefficients as follows;

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}$$

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}$$

$$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}$$

$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}$$

The wavelet function coefficient values are:

$g_0 = h_3$

$g_1 = -h_2$

$g_2 = h_1$

$g_3 = -h_0$

Each step of the wavelet transform applies the wavelet function to the input data. If the original data set has N values, the wavelet function will be applied to calculate N/2 differences (reflecting change in the data). In the ordered wavelet transform the wavelet values are stored in the upper half of the N element input vector. The scaling and wavelet functions are calculated by taking the inner product of the coefficients and four data values.

3.2.3 Bi-orthogonal Wavelets

A bi-orthogonal wavelet is a wavelet where the associated wavelet transform is invertible but not necessarily orthogonal. Bi-orthogonal wavelets allow more freedom than orthogonal wavelets an example of which is the possibility to construct symmetric wavelet functions. [8] Bi-orthogonal wavelets make use of two scaling functions $\emptyset, \tilde{\emptyset}$, which may generate different multi-resolution analyses, and accordingly two different wavelet functions $\psi, \tilde{\psi}$. So the numbers *M* and *N* of coefficients in the scaling sequences a, $\tilde{a}$ may differ. The scaling sequences must satisfy the following bi-orthogonality condition

$$\sum_{n \in Z} a_n \, \tilde{a}_{n+2m} = 2. \; \delta_{m,0}$$

Then the wavelet sequences can be determined as

$$b_n = (-1)^n \tilde{a}_{M-1-n} , n = 0, \ldots, M-1 \text{ and}$$

$$\tilde{b}_n = (-1)^n a_{M-1-n} , n = 0, \ldots, N-1.$$

3.3 Applications

Discrete Wavelet Transform finds applications in Signal De-noising and Data Compression. DWT is a form that is appropriate to discrete data such as individual points or pixels in an image. On account of this property DWT is often used in Image compression schemes.

3.4 DWT in Image Compression

DWT is used in a number of Image compression schemes among which some of the notable ones are EZW (Embedded zero-tree wavelet), SPIHT and JPEG 2000. (Joint-Photographic Experts Group) The EZW used DWT for the decomposition of an image at each level and scans the wavelet coefficients in each sub-band in a zigzag manner. SPIHT is a highly refined version of the EZW and it gives higher image quality with higher PSNR and compression ratios as compared to the EZW.

The JPEG 2000 standard does not have the limitations of the previous versions of JPEG because it has incorporated the use of wavelet technology. The JPEG 2000 uses Wavelet transform and Arithmetic Coding. For lossy compression, it uses the Daubechies (9,7) filters and Daubechies (5,3) filters for lossless compression.

The discrete wavelet transform (DWT) is a form appropriate to discrete data such as individual points or pixels in an image. The DWT runs a high pass and a low pass filter over the image in one dimension which could be either horizontal or vertical. This produces a low pass and a high pass version of the data. These results are down sampled, yielding two sub-bands, each of which is one half the size of the input stream. The result is a new image comprising of a high-pass and low-pass sub-band. For a multidimensional signal such as in image, this procedure of filtering and sampling is repeated in each dimension. [8] The vertical and horizontal transformations break up the image in four distinct sub-bands. The wavelet coefficients that represent the fine details are LH, HH and HL. LL represents the lower frequency component of the image. Figure 3.1 explains this process

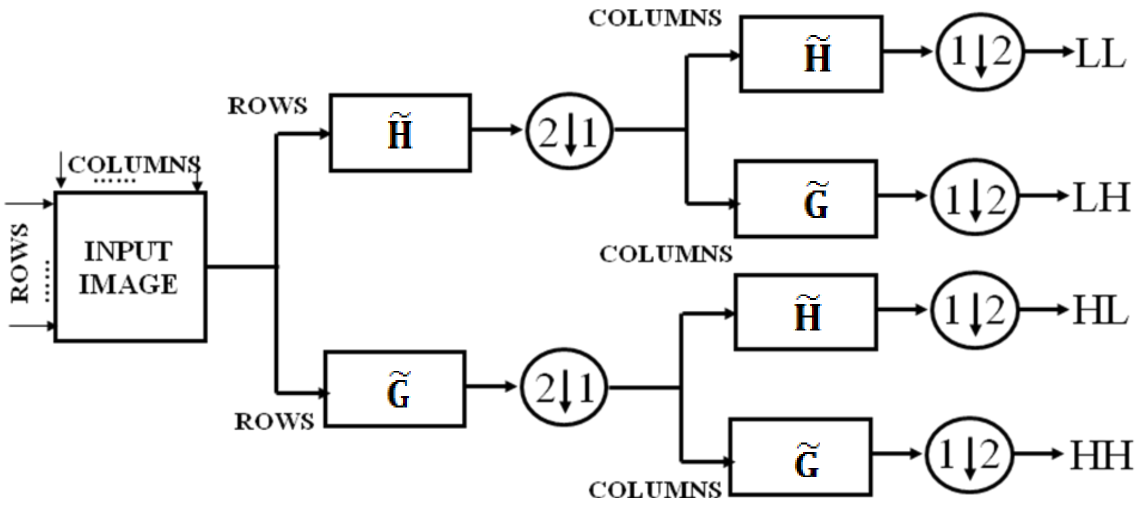
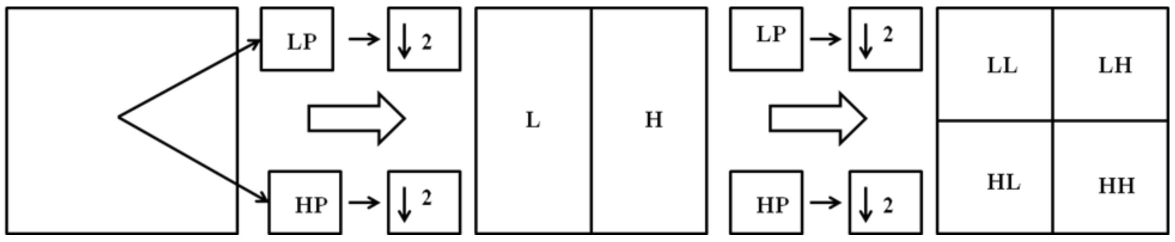


Figure 3.1 2-D Discrete Wavelet Transform of an image [9]

Figure 3.2 explains a one-level wavelet transform over an image by two one-dimensional passes.



(a) (b) (c)

Figure 3.2 A 1-level wavelet built by two one-dimensional passes: (a) original image (b) horizontal pass, and (c) vertical pass. [8]

For the next level calculation, the horizontal and vertical transformations are carried out on the LL sub-band from the previous level. In Figure 3.3(a), 2-level DWT is shown. Similarly for a 3-level DWT the transformations will be carried out on LL sub-band from 2-level transform. Figure 3.3(b) shows the results of a 3-level transform.

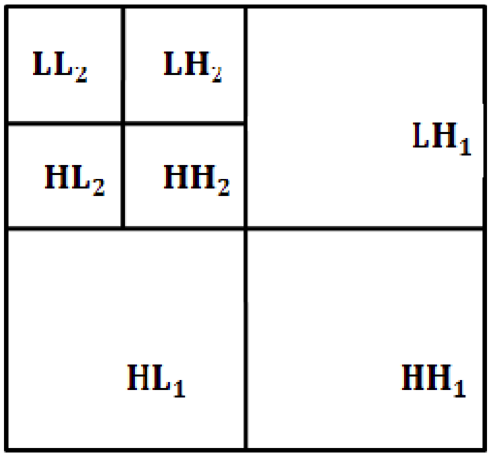


Fig 3.3(a) 2-level DWT

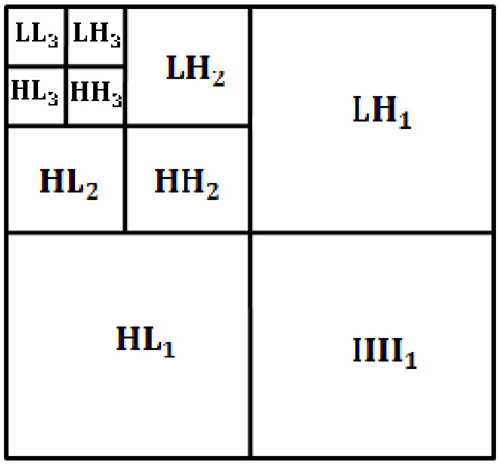


Fig 3.3(b) 3-level DWT

This process can go on for 'n' number of times with each level giving finer and finer details of an image. Figure 3.4(a) and Figure 3.3(b) help visualize the effects of the transform on a sample MRI image.

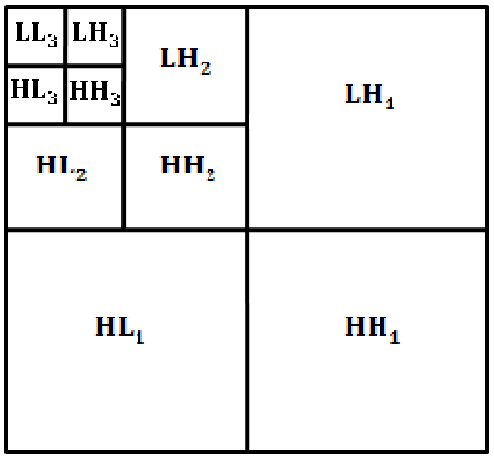


Fig 3.4(a) 3-level DWT effect

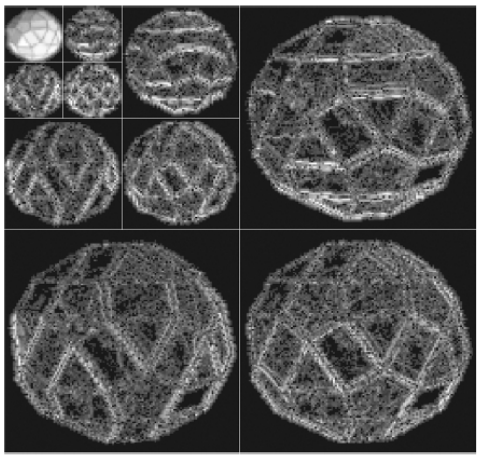


Fig 3.4(b) Sample medical image

CHAPTER 4

FIELD-PROGRAMMABLE GATE ARRAY

4.1 Introduction

Field Programmable Array (FPGA) is a reconfigurable integrated circuit. It is called field programmable because it is an integrated circuit, designed in such a way that it can be configured by the customer or designer after it has been manufactured. It contains logic blocks and programmable resources to implement a wide number of hardware functions. These logic blocks are depicted in Figure 4.1.

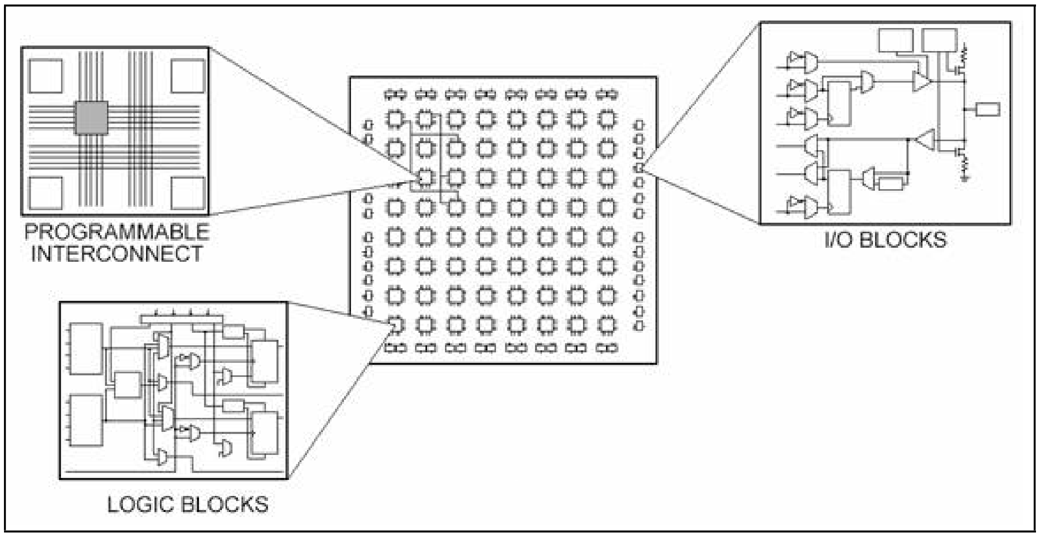


Figure 4.1 Basic building blocks of FPGA [11]

FPGAs are configured using a hardware description language (HDL), Verilog HDL or VHDL (VHSIC HDL which is the abbreviation for Very-high-speed Integrated Circuit HDL). They can perform digital as well as analog functions.

Today most industries around the globe have adopted FPGA based applications due to the fact that FPGAs offer a combination of the best characteristics of ASICs and processor-based systems. FPGAs provide high speed and reliability but unlike ASIC design, they do not require large memory space.

FPGAs contain programmable logic components called ‘logic blocks’ which are wired together by reconfigurable interconnects. These logic blocks can be configured according to the users requirements to perform complex combinational computations or simple logic functions such as AND, OR and XOR. The logic blocks in most FPGAs also include memory elements, which may be simple flip-flops or more complete memory blocks such as Block RAMs. [9]

4.2 Industrial Scope of FPGA

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). Since the invention of the first commercially available FPGA by Xilinx in 1985, the worldwide FPGA market has grown from $14 million in 1987 to an estimated $2.75 billion in 2010.

The current FPGA market leaders are Xilinx Inc and Altera, which control 80% of the market. Xilinx reportedly represents over 50% of the market share. According to Moshe Gavrielov, the new CEO and president of Xilinx, “The programmable market is worth $4bn and is estimated to be worth $5bn in 2011, but the ‘opportunity’ market for programmable devices is $14bn” [10].

The first FPGA was invented by Xilinx in 1984 and the early devices were simple logic chips. Today they find applications in most signal processing systems and control applications, and are now rapidly replacing custom Application-Specific Integrated Circuits (ASICs).

4.3 Overview of Spartan 3E Family

The Spartan-3 family architecture consists of five fundamental programmable functional elements [11]:

- Configurable Logic Blocks (CLBs) contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.

- Input/output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. It supports a variety of signal standards, including four high-performance differential standards.

- Block RAM (BRAM) provides data storage in the form of 18-Kbit dual-port blocks.

- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.

- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device.

The Spartan-3E family features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

4.3 Overview of Virtex-4 Family

The Virtex-4 family is divided in three sub-families which give very high performance logic applications solutions, for DSP applications and embedded platform applications. Following are some of the basic blocks of the Virtex-4 family [12].

CLBs, Slices, and LUTs

Virtex-4 FPGA has number of slices ranging from 6,144 to 63,168 and maximum distributed RAM range from 96 Kb to 987 Kb. The configurable logic blocks (CLB) provide combinatorial and synchronous logic as well as distributed memory and shift register capability.

500 MHz DSP Slices

Cascadable embedded DSP slices with 18-bit x 18-bit dedicated multipliers, integrated Adder, and 48-bit accumulator. These DSP Slices give up to 100% speed improvement over previous generation devices.

Clock Distribution

Each Virtex-4 FPGA provides up to 20 Digital Clock Manager (DCM) modules offering flexible frequency synthesis, improved maximum input/output frequency and reduced output jitter. It gives self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication/division, and coarse/fine-grained clock phase shifting.

Block RAM

Every Virtex-4 FPGA has between 48 and 552 dual-port block RAMs, each storing 18 Kbits. Each block RAM has two completely independent ports that share nothing but the stored data.

Input/output

The number of I/O varies from 320 to 960 depending on device and package size. Each I/O pin is configurable and can comply with a large number of standards, using up to 2.5V.

CHAPTER 5

SET PARTITIONING IN HIERARCHICAL TREES

5.1 Introduction

The SPIHT (Set-Partitioning in Hierarchical Trees) algorithm was developed in 1996 by Amir Said and William Pearlman and is an advancement of the Embedded Zerotree Wavelet algorithm (EZW) which was developed by J.M. Shapiro in 1993 [13]. The roots of SPIHT may be embedded in the EZW algorithm however it is not a simple extension of the later, and has proven to be a benchmark in the field of image compression algorithms.

SPIHT brought about a revolution in the field of image compression because it broke the trend to complex compression algorithms. Previous research works involved using sophisticated vector quantization in various attempts to improve coding schemes. SPIHT achieved superior results compared to vector quantization while employing the simplest technique of uniform scalar quantization. Therefore, it is much easier to design fast SPIHT codecs.

SPIHT is a wavelet-based image compression coder [1]. It is a simple and efficient algorithm with many unique and desirable properties such as low complexity, low memory usage and offering progressive transmission in both lossy and lossless coding. First the wavelet coefficients of the image are computed using the discrete wavelet transform and then information about these coefficients is transmitted [14] SPIHT algorithm codes the pixels according to their significance.

5.2 SPIHT Coder

SPIHT makes use of three lists namely

List of Significant Pixels (LSP)

List of Insignificant Pixels (LIP)

List of Insignificant Sets (LIS)

Some other important definitions which are used in the SPIHT coder are as follows;

O (i,j): set of coordinates of all offspring of node (i,j).
D (i,j): set of coordinates of all descendants of node (i,j).
H (i,j): set of all tree roots (nodes in the highest pyramid level).
L (i,j): **D** (i,j) – **O**(i,j) (all descendents except the offspring)

Wavelet coefficients are compared with a threshold value at each level, and manipulated between the lists LIP, LIS and LSP. There are also two types of entries used.
Type A entry: D(i,j)
Type B entry: L(i,j)

If a pixel is significant its magnitude must be greater to or equal to the current threshold. An insignificant pixel value falls beneath the current threshold level. The following explanation clarifies this significant criterion.

For a pixel: $|c(i,j)| \geq 2^n \rightarrow$ Significant 2^n

For a set S: $\max|c(i,j)| \geq 2^n \rightarrow$ Significant

Similarly;
For a pixel: $|c(i,j)| < 2^n \rightarrow$ Insignificant

For a set S: $\max|c(i,j)| < 2^n \rightarrow$ Insignificant

Where n is the number of bits of the largest coefficient

The SPIHT coder follows a number of steps in a predefined form at every level of operation. There are two important steps in SPIHT namely, The Sorting Pass and the Refinement Pass. The number of Sorting Passes is always one less than the number of Refinement Passes. In the Sorting Pass, each pixel is compared with the threshold. If the magnitude of a pixel exceeds or equals the current threshold it is termed as significant. A pixel is termed as insignificant if its magnitude falls beneath the current threshold. An insignificant set can be one of two types of sets. The set H contains all the pixels in the last level of the wavelet transform that was performed, including the coarse and detail coefficients [14].
After each level the threshold is decreased by a factor of $2^{n-1}$and the process is repeated.

After the initialization, the algorithm takes two stages for each level of threshold. First the sorting pass is carried out in which the lists are organized. Following the sorting pass, the refinement pass is carried out which does the actual progressive coding transmission. [15] The result is in the form of a bit stream. The detailed scheme of the complete algorithm is shown in Fig 5.1

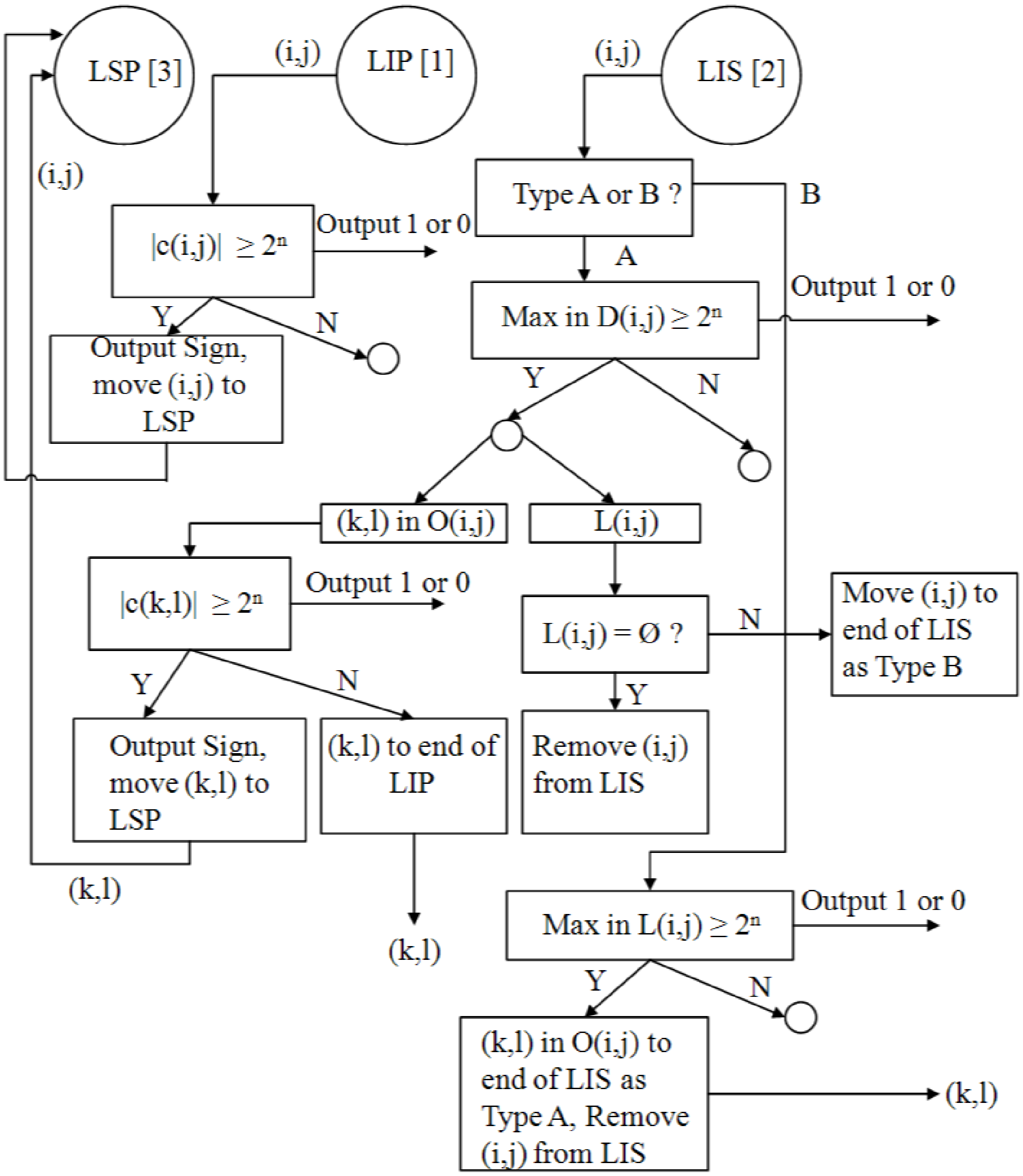


Figure 5.1 SPIHT coding workflow [16]

5.3 Properties

SPIHT is the remarkable algorithm which has displayed all of the below discussed properties. There may be other image compression algorithms which give better performance in one area but the important consideration here is that SPIHT wins in the remaining criteria. The characteristics of SPIHT are discussed in detail in the following section. [16]

- Shows good image quality and high PSNR, especially for color images. SPIHT exploits the properties of the wavelet-transformed images to increase its efficiency.
- Progressive Image Transmission is an inherent quality of the SPIHT algorithm. This means that the quality at the decoder end improves as it receives more information about the original image.
- Optimized Embedded coding; this is a special characteristic which enables compression of a single image in different sizes to suit the individual needs of different users. By using this technique an image need not be compressed each time for a different user and gives an image quality comparable or even superior to sophisticated non-embedded encoders.
- Symmetric encoding and decoding; the time taken for encoding is nearly equal to the time taken for decoding an image sequence.
- It can be used for lossless compression and on account of this property can be effectively applied to medical imaging or areas where degradation or loss of information is a critical issue.
- SPIHT can be efficiently combined with error protection.
- SPIHT offers Multi-resolution scalability which means that the encoder and decoder track the resolution of bits automatically.

5.4 SPIHT Coding Engine

The SPIHT coding engine performs the following four tasks in the given order, each of which is explained in the following sections.

1. Initialization
2. Sorting Pass
3. Refinement Pass
4. Quantization-step update

5.4.1 Initialization

It initializes the value of '*n*' for testing significance of pixels and constructing significance map. The LSP is set as an empty list. The LIS is initialized to maintain all pixels in the low pass sub band that have descendents and hence act as roots of spatial trees. All these pixels are assigned to be of type A. LIP is initializing to contain all pixels in low pass pixels. The threshold with which all the pixels are compared is computed using the formula 2^n where '*n'* can be calculated by the formula in step 1.

1. ***n*** = $\lfloor log_2(\max\{c(i,j)\})\rfloor$ where c(i,j *)* is the coefficient at position (i, j) in the image.
2. LIP = All elements in H
3. LSP = Empty
4. LIS = **D**'s or descendants of Roots

5.4.2 Sorting Pass

In the Sorting Pass, the magnitude of pixels is manipulated between the three lists LIP, LIS and LSP after being compared with the threshold value. In this pass, elements of the LIP may be moved to the LSP. Sets contained in LIS are broken down into the relevant tree type i.e. A-type or B-type and their offspring may be moved to LIP or LSP according to their significance. Each entry of the LIP is tested for significance with respect to the threshold. If it passes the significance criterion, a 1 is transmitted followed by a sign bit representing sign of that pixel and the pixel coordinates are moved to LSP. If the entry is insignificant then a 0 is transmitted. The Sorting Pass process has been summarized in a step-by-step form as follows:

1. Process LIP.

a) For each coefficient (i,j) in LIP, Sn (i,j) is output where Sn (i,j) =1 when max |c(i,j)| >= 2n or Sn (i,j) = 0 for other.

b) If Sn (i,j) =1 ,sign of coefficient (i,j): 0/1 is output and (i,j) is moved to the LSP.

2. Process LIS.

a) For each entry (i,j) in LIS and if the entry is of type D then output Sn(D(i,j)).

i) If Sn(D(i,j)) = 1 then for each (k,l) ∊ O(i,j) output Sn(k,l).

ii) If Sn(k,l) = 1, then add (k,l) to the LSP and output sign of coefficient: 0/1 .

iii) If Sn(k,l)=0, then add (k,l) to the end of the LIP.

b) If type L then output Sn(L(i,j)).

i) If Sn(L(i,j)) =1 then add each (k,l) ∊ O(i,j) to the end of the LIS as an entry of type D and remove (i,j) from the LIS. Figure 5.2 shows the workflow of the sorting pass.

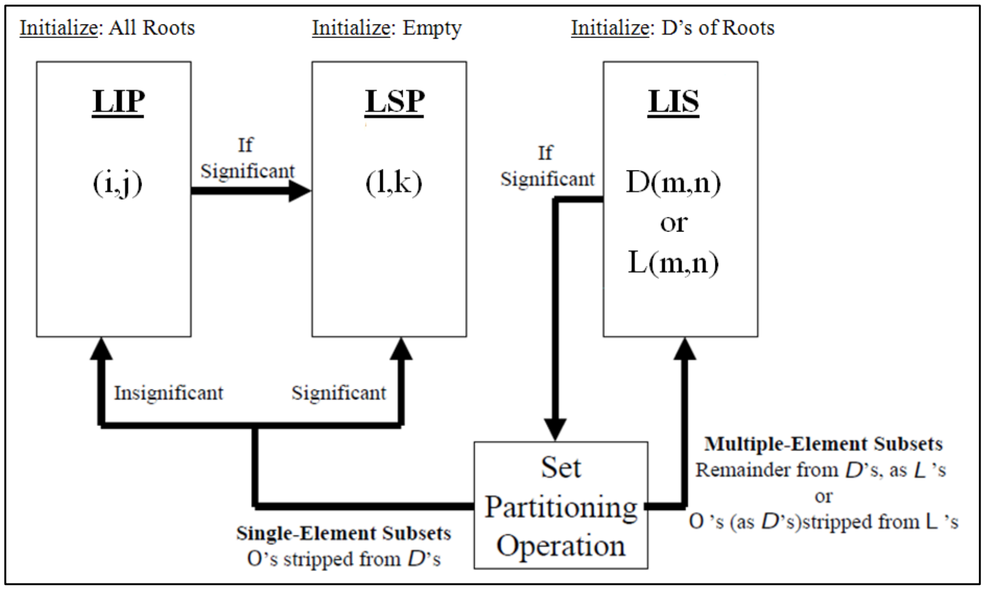


Figure 5.2 Sorting Pass [17]

5.4.3 Refinement pass

The refinement pass follows the sorting pass and it processes the entries in the LSP from the previous sorting pass. The entries to the LSP from the current sorting pass are ignored and the nth MSB of the magnitude of each entry from the previous pass is transmitted to the decoder. It is important to note here that since refinement pass works upon entries from the previous sorting pass, no bits would be transmitted at the end of the first sorting pass because the LSP would contain no pixels prior to the current sorting pass. Refinement pass plays the main role in decoding or reconstruction as it is responsible for transmitting the bit corresponding to the current magnitude threshold for each entry in the LSP, which was not added in the previous sorting pass. The refinement pass process has been summarized in the following two steps.

1. Process LSP.
2. For each element (i,j) in LSP except those just added above in the sorting pass the nth most significant bit of coefficient is output. Figure 5.3 explains the refinement pass.

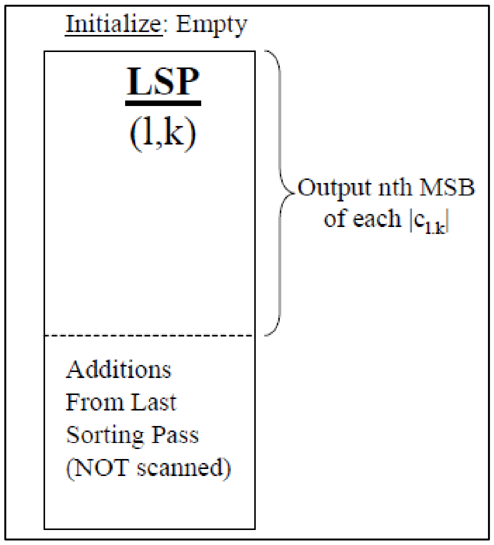

Initialize: Empty
LSP
(l,k)
Output nth MSB of each $|c_{l,k}|$
Additions From Last Sorting Pass (NOT scanned)

Figure 5.3 Refinement Pass [17]

5.4.4 Quantization Step Update

The quantization step updates simply decrease the threshold by decreases the value of N. This step is repeated after the sorting and refinement pass for a certain threshold has completed. The value of N is decremented by 1 subsequently decreasing the threshold by a factor of 2^{n-1} .The algorithm then returns to the sorting pass at step 2 and continues in the defined order. The summary of the quantization step update is given in the following two steps.

1. Decrement n by 1.
2. Then go back to the Significance Map Encoding Step (Sorting Pass).

5.5 Decoder

In addition to performing the pre-defined job of decoding, the decoder also performs the task of updating the reconstructed image. For the value of *n* when a coordinate is moved to the LSP, the condition in equation 5.5.1 is known;

$$2^n \leq | C\ i, j| < 2^n \text{ ……………………………….. Eq (5.5.1)}$$

So, the decoder uses that information, plus the sign bit that is input just after the insertion in the LSP, to set the value of $C_{i,j}$ using Equation 5.5.2;

$$C\ i, j = \pm 1.5 * 2^n \text{ ………………………………Eq (5.5.2)}$$

Similarly, during the refinement pass the decoder adds or subtracts 2n - 1 to $C_{i,\ j}$ when it inputs the bits of the binary representation of $| C_{i,j} |$.

5.6 Basic Algorithm

The following gives a step-by step implementation of the SPIHT encoder. [18]

1. Compute the threshold. Initialize LIP to all the root node coefficients. LIS to all the trees (assign Type-D to them), LSP to an empty set.

2. Check the significance of all coefficients in LIP.
    - If significant, output 1, followed by a sign bit and move it to the LSP.
    - If not significant, output 0.

3. Check the significance of all the tress in LIS according to tree type.

a. For a tree of Type-D
    - If it is significant, output 1, and code its children
        - ➢ If a child is significant output 1, then a sign bit and add it to LSP.
        - ➢ If a child is insignificant, output 0 and add to the end of LIP.
        - ➢ If the children have descendants move the tree to the end of LIS as Type L, otherwise remove it from LIS.
    - If it is insignificant, output 0.

b. For a tree of Type-L
    - If it is significant, output 1, add each of the children to the end of LIS as an entry of Type D and remove the parent tree from the LIS.
    - If it is insignificant, output 0.

CHAPTER 6

OUR DESIGN ARCHITECTURE

6.1 Introduction

This chapter pertains to our design architecture and how we have integrated software and hardware platforms to achieve our goal. Our design has been split in two halves wherein half of the work is implemented on software and the remaining half on hardware. Our work comprises of four phases. Phases 1 and 4 have been incorporated in the software architecture and Phases 2 and 3 on hardware platform. Our software architecture has been designed on MATLAB and hardware architecture is implemented on FPGA.

The DWT and Inverse DWT have been carried out in Phase 1 and Phase 4 respectively while the SPIHT encoder is designed on Phase 2 and SPIHT decoder is designed in Phase 3. In Phase 1, the DWT of the sample image is computed which yields the wavelet coefficients. In Phase 2, the wavelet coefficients are given to the SPIHT encoder which encodes the wavelet coefficients. In Phase 3, the coded wavelets are decoded in the decoder block. In Phase 4, the inverse DWT is carried out on the decoded wavelets to reconstruct the original image. Figure 6.1 explains our design workflow.

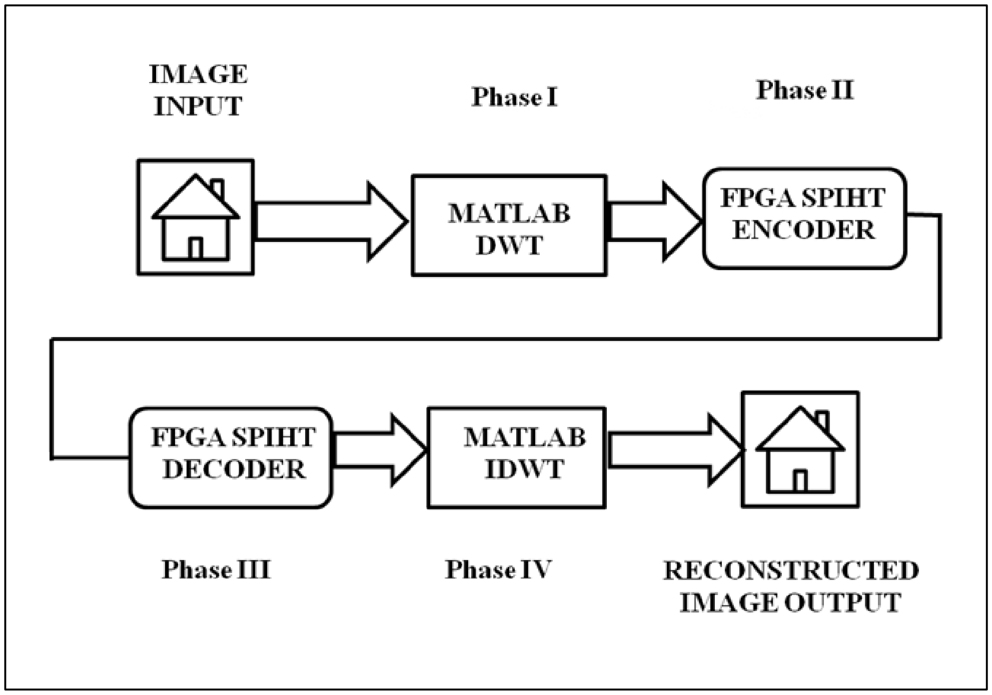


Figure 6.1 Our Design Architecture

6.2 Software architecture

Discrete Wavelet Transform using HDL requires lengthy coding and a long verification exercise. For this reason, we employed the help of a high-level language i.e. MATLAB to compute the wavelet coefficients of the sample image. We have used Daubechies wavelets to compute four-level 2-D DWT of our image. We have used a simple 256 x 256 sample image to implement the SPIHT algorithm as shown in Fig 6.3(a). The sample image is read onto a MATLAB code which computes the 4-level DWT on the image using db-1 filters. Following a DWT, the image is divided in to Spatial Orientation Trees which are described in detail later in this section. The Spatial Orientation Trees are visible in Fig 6.3(b) which has been obtained using MATLAB wavelet toolbox. These trees are then converted to hexadecimal form and given as input to the SPIHT Encoder. Figure 6.2 shows the workflow of Phase 1, implemented on software.

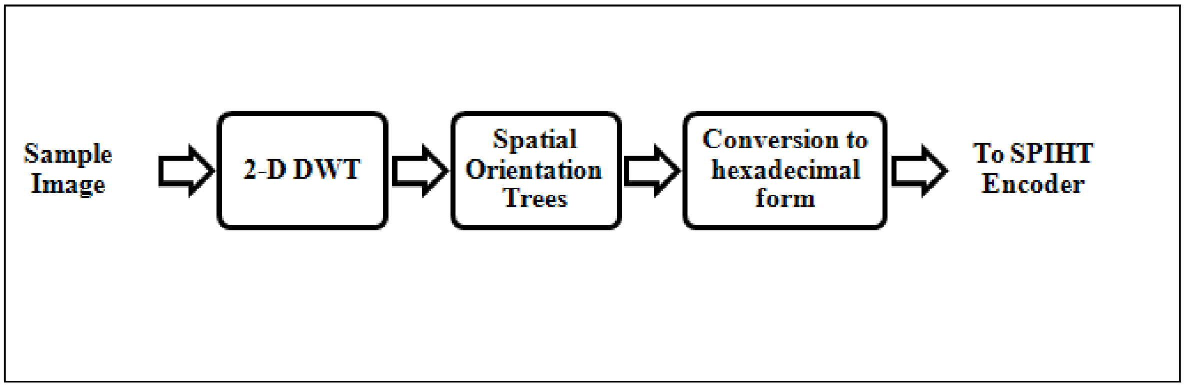
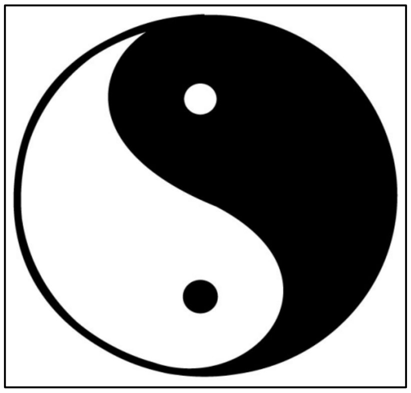


Figure 6.2 Workflow in Phase 1 on Software

The sample image and its decomposition have been shown in the figures below.



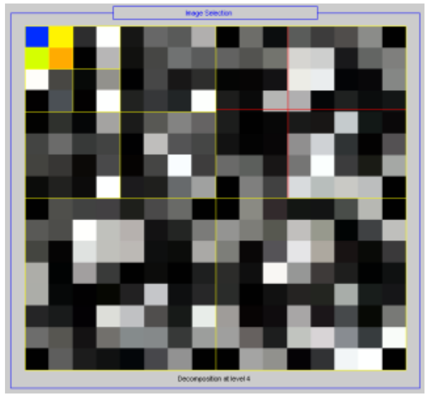


Figure 6.3(a) Sample Image

Figure 6.3(b) 4-level Decomposition

Following a DWT, the requirement of SPIHT scheme is for the image matrix to be split into Spatial Orientation Trees. The wavelet coefficients are then compounded together in a single matrix and converted to hexadecimal form so they can be read onto the FPGA.

The Spatial Orientation trees basically define the parent-children relationship between the pixels in an image as depicted in Figures 6.4(a) and 6.4(b). In 6.4(a), the relationship between parent node and offspring is shown using arrows. Each node in a tree corresponds to an individual pixel. The offspring of a pixel are the four pixels in the same spatial location at the next finer scale of the wavelet. SPIHT uses this relationship between nodes and their descendants when coding the wavelets.

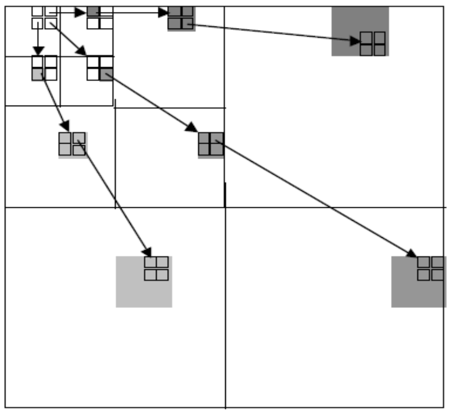


Figure 6.4(a) Spatial Orientation Trees [8]

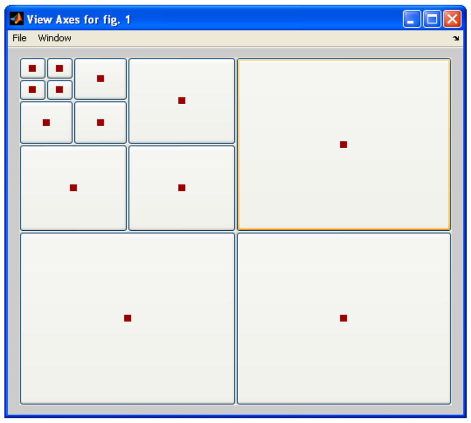


Figure 6.4(b) Tree Axes

The last portion of our work is implemented on software which is the final Phase 4 of our design. In it the decoded wavelets from the FPGA decoder serve as input. After the decomposed wavelets are coded by the FPGA coder and consequently recovered by the decoder they have to be given to MATLAB again for the Inverse DWT to be performed. The IDWT results in the reconstruction of the original image. Since a 4-level DWT was performed on the original image in Phase 1, we performed a 4-level Inverse DWT on the decoded wavelets for the recovery of the image as depicted in Figure 6.5

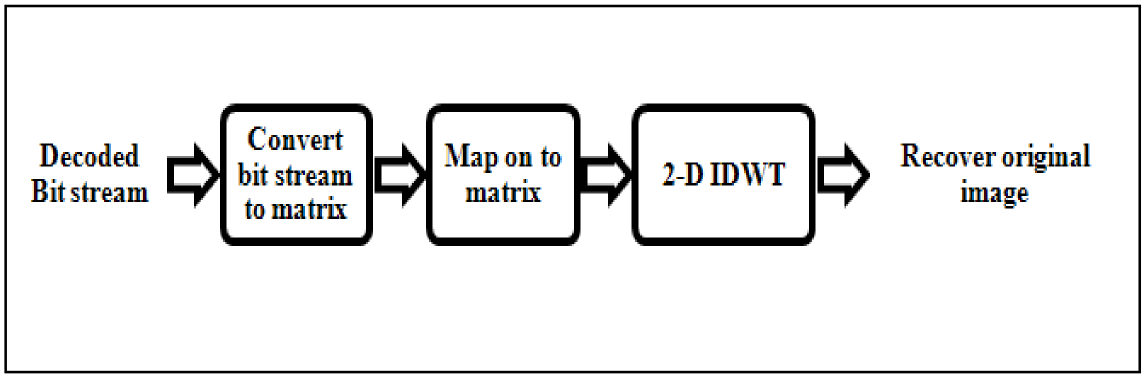


Figure 6.5 Workflow in Phase 4 on software

6.3 Hardware architecture

As discussed in Section 6.1, our design architecture comprises of four phases amongst which Phase II and III i.e. the SPIHT encoder and decoder sections have been implemented on FPGA. Our design has four Verilog modules, three of which pertain to the encoder block and a single module for the decoder block. In Figure 6.5, these four modules have been combined in a block diagram to explain the workflow of our hardware design.

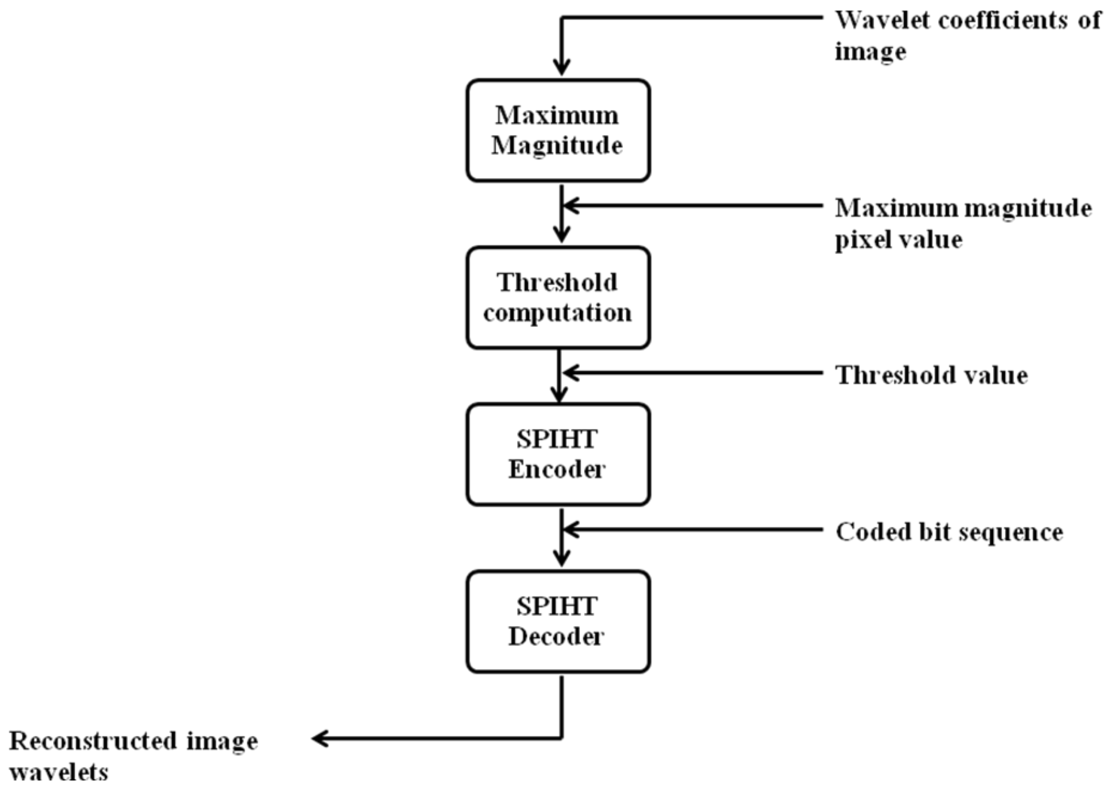


Figure 6.6 Hardware Architecture

In the following sections we have explained the functionality and implementation details of each module individually.

6.3.1 Maximum Magnitude

The software portion of our design calculates the 16 x 16 wavelet array of the sample image. This array comprises of all elements represented in 32-bit hexadecimal format. The wavelet array is then given as input to the first module of the hardware which returns the maximum magnitude pixel after scanning all the elements in the image array. The target device for this module is a XC3S500E device from Spartan 3E family. The module uses 67 4-input LUTs and fits into 35 slices, thus utilizing 3% memory resources of the Spartan 3E device. It operates at a clock rate of 74 MHz and gives a throughput of 2368 Mbs.

6.3.2 Threshold computation

In this module, the maximum magnitude pixel value from Maximum Module is treated as input. First the log base 2 of this value is computed and then the floor of the log value is calculated. This floor value is then used to compute the threshold for Sorting Pass 1 of the SPIHT encoder. The variables calculated in this module will be used in the following equation,

$$n = \lfloor \log_2 (\text{maximum magnitude}) \rfloor$$

In the above equation, the maximum magnitude value is taken from the previous module, and the output value 'n' will be used to compute the threshold 'T' using the following formula $T = 2^n$. This value serves as the initial threshold for the first sorting pass in the encoder and with each preceding sorting pass, the threshold value decreases by a factor of $n-1$. The target device for this module is XC3S100E from Spartan 3E Family. The module uses 133 4-input LUTs and occupies 72 slices, thus utilizing 7% memory resources of the Spartan 3E device. It operates at a clock rate of 67 MHz and gives a throughput of 2144 Mbps.

6.3.3 SPIHT Encoder

In the previous module, the threshold has already been calculated. This will be treated as the initial value in Sorting Pass 1. The threshold is decremented in each sorting pass by a ratio of 2^{n-1}. The encoder then goes through a series of Sorting Passes and Refinement Passes. As the encoder steps through the algorithm it inserts or deletes pixels from the three lists. All of the information required to keep track of the lists is output to the decoder. In this way the decoder maintains and generates an identical list order as the encoder. For the decoder to reproduce the steps taken by the encoder, the output statements in the encoder's algorithm can be replaced with input for the decoder's algorithm. [10] In our design, we have calculated till 5th Sorting Pass. At the end of the 5th Pass we have a coded bit stream output which will be given to the decoder. Figure 6.6 depicts the encoder workflow on FPGA.

The encoder module has been implemented on XC4VLX80 Virtex-4 Device. The slice usage is 21327 which are 59 % of the total available resources. The reasons for using a high density device is the large number of slice LUTs used to implement the design and the extensive data manipulation that takes place in the encoder module. The entire module uses 40389 LUTs which rounds up to 56 % usage of available resources. It operates at a clock rate of 3.143 MHz and gives a threshold of 6436 Mbps.

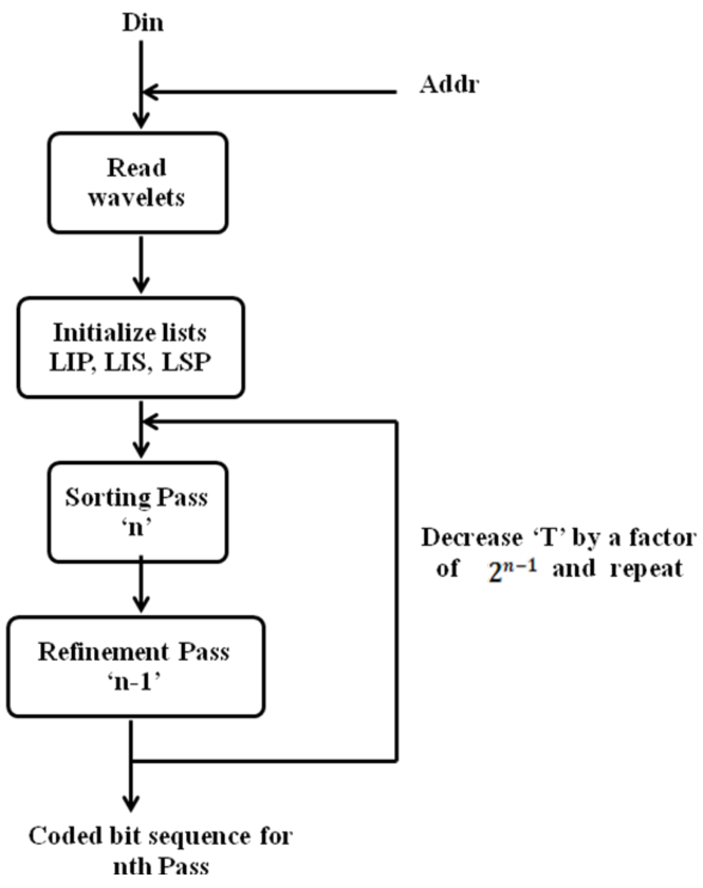

Din
Addr
Read wavelets
Initialize lists LIP, LIS, LSP
Sorting Pass 'n'
Refinement Pass 'n-1'
Decrease 'T' by a factor of 2^{n-1} and repeat
Coded bit sequence for nth Pass

Figure 6.7 SPIHT Encoder Workflow

6.3.4 SPIHT Decoder

The decoder follows a list order which is identical to the one used by the encoder. The coded bit stream from the encoder contains the bits from the sorting pass and refinement passes. These are given as input to the decoder. Also, initially included in the decoder module is a blank image array. Once the coded bit stream from the encoder is given to the decoder; the pixels are mapped on to the image matrix according to their location. Next, the correction bits are computed and these are used to overwrite on the image matrix. This overwritten result is a closer version of the original DWT. If there are more bit sequences in pipeline, this process of mapping and overwriting will be repeated. When all the bits from the encoder output have been mapped upon the image matrix, we have a reconstruction of the original wavelet transform. The decoder maintains an identical list order as the encoder. Figure 6.6 elaborates the workflow of the SPIHT decoder.

The target platform for the decoder module is XC3S100E device from Spartan-E Family. The decoder module utilizes 112 4-input LUTs and 62 slices which comprise of 6% of the available resources. It operates at a clock rate of 222 MHz and gives a throughput of 1776 Mbps.

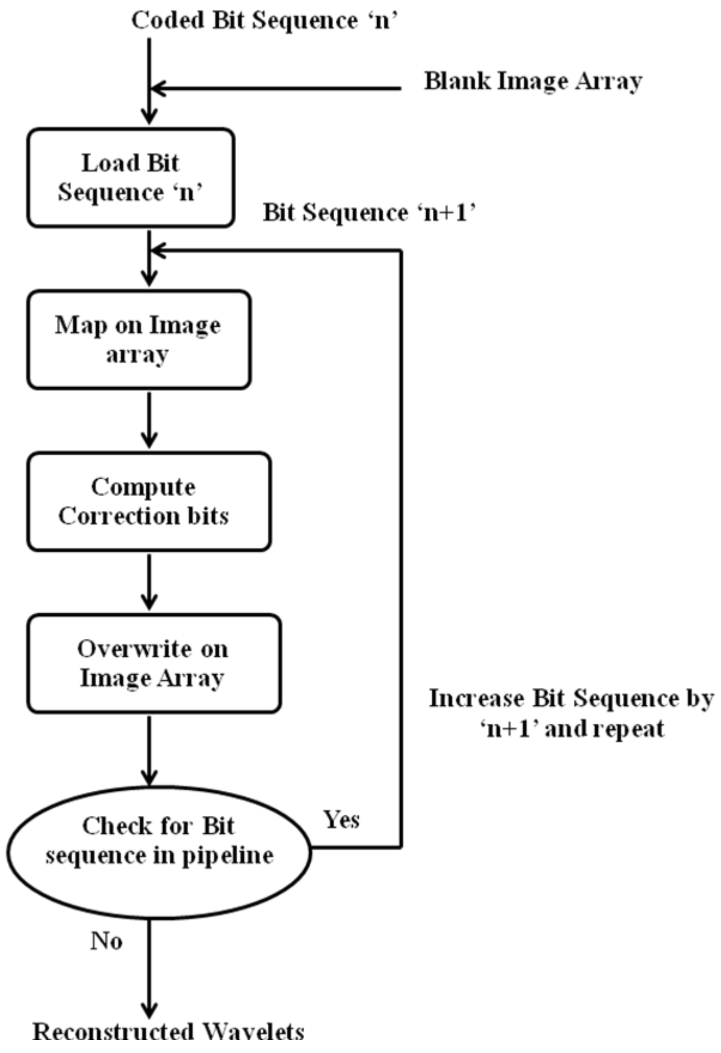


Figure 6.8 SPIHT Decoder workflow

CHAPTER 7

IMPLEMENTATION RESULTS AND COMPARISIONS

7.1 Introduction

In this chapter we have compiled the implementation results of our hardware architecture and compared them with previous known implementations. As discussed in Chapter 6, our hardware design architecture comprises of four modules namely Maximum magnitude, Threshold, SPIHT encoder and SPIHT decoder. We have tabulated the results of each module in terms of clock rate and FPGA area utilization.

Furthermore we have compared our results with previous implementations of SPIHT on FPGA. As we have observed that previously SPIHT has been implemented on hardware by taking different kinds of approach. Researchers have made a number of criteria, the centre of their focus. Some projects have been implemented while taking device performance, clock rate, system frequency and throughput in consideration while others have been compared and optimized in terms of PSNR, Image quality, compression ratios and bits per pixel usage. Some algorithms have also been modified in terms of resource utilization and speed. A common factor that we have so far observed is that the SPIHT algorithm has been implemented either entirely on hardware or entirely on software. Our work is different in the sense that we have tried to integrate the individual capabilities of both hardware and software to achieve optimized results. We have studied a number of research works and compared our results individually with the common parameters found in these works.

Our proposed designs have been implemented using Xilinx ISE Foundation 11.3 tool and have successfully passed synthesis and Place and Route processes. We have successfully tested each module for correct results using ISE Simulator. In Section 7.3 we have compared our simulation time with that of previous results. In Section 7.4 we have compared our FPGA area utilization with other projects.

7.2 Implementation Results

Our hardware design is build on four modules in which three modules have been implemented on Spartan 3A device and one module on Virtex 4 device. Since our encoder module is most computation extensive, we decided to implement it on a higher density Virtex-4 device which offers faster processing speed and higher resources as compared to the cost effective Spartan 3E device. We have carried out the post-synthesis process on each of our modules and implemented the design successfully. In the following sections we have tabulated the synthesis results of each module in their order of execution.

7.2.1 Maximum Magnitude Module

The target device is XC3S500E from Spartan 3E Family. It computes the maximum magnitude pixel from the wavelet coefficient image array, computed in MATLAB and given to the Verilog program in hexadecimal form. This module scans the array for the maximum value pixel and returns the output. It operates at a clock rate of 74 MHz and gives a throughput of 148 MPixels/sec while occupying an FPGA area of 3% in terms of slices. Table 6.1 gives the implementation results of Maximum Magnitude module.

Table 7.1 Implementation Results of Maximum Magnitude module

Target FPGA Device	**XC3S500E-5**
Module Name	**Maximum Magnitude**
Clock Rate	74 MHz
Throughput	148 MPixels/sec
Device Utilization	**Percentage of total available resources**
Number of Slices	3 %
Number of 4 input LUTs	3 %
Number of bonded IOBs	98%

7.2.2 Threshold Module

The target device is XC3S100E from Spartan 3E Family. It takes the maximum pixel value from the maximum magnitude module and performs two functions on it. First the log base 2 of the maximum value is computed and then the floor of the previous result is computed. The final result is used to calculate the initial threshold. This module performs a small function but involves extensive arithmetic computations. The module operates at a clock rate of 67 MHz and fits into 7% of the total available slices. It gives a throughput of 268 MPixels/sec. Table 7.2 tabulates the synthesis results of the threshold module

Table 7.2 Implementation Results of Threshold module

Target FPGA Device	**XC3S100E-5**
Module Name	**Threshold**
Clock Rate	67 MHz
Throughput	268 MPixels/sec
Device Utilization	**Percentage of total available resources**
Number of Slices	7 %
Number of 4 input LUTs	6 %
Number of bonded IOBs	88 %

7.2.3 SPIHT Encoder

The target device is XC4VLX80 from Virtex-4 Family. The encoder module involves most computations as compared to the other hardware modules. It goes through a pipelined process of sorting and refinement passes whereby each pixel is scanned and manipulated individually. The coder operates at a clock rate of 98 MHz and gives a throughput of 294 MPixels/sec while using 59% of available slices. Results of the encoder block are tabulated in Table 7.3.

Table 7.3 Implementation Results of Encoder module

Target FPGA Device	**XC4VLX80**
Module Name	**SPIHT Encoder**
Clock Rate	98 MHz
Throughput	294 MPixels/sec
Device Utilization	**Percentage of total available resources**
Number of Slices	59%
Number of Slice Flip-Flops	3%
Number of 4 input LUTs	56%
Number of IOs	200

7.2.4 SPIHT Decoder

The target platform for decoder module is XC3S100E device from Spartan-E Family. The coded bit sequence from the encoder is processed by the decoder and individual mapping of pixels on the wavelet reconstruction array takes place. The decoder module works at a clock rate of 69 MHz and gives a throughput of 276 MPixels/sec while occupying 6% of the available area resources. Table 6.4 shows the implementation results of the decoder module.

Table 7.4 Implementation Results of Decoder module

Target FPGA Device	**XC3S100E-5**
Module Name	**SPIHT Decoder**
Clock Rate	69 MHz
Throughput	276 MPixels/sec
Device Utilization	**Percentage of total available resources**
Number of Slices	6 %
Number of 4 input LUTs	5 %
Number of bonded IOBs	37 %

7.3 Time Comparison

Our Maximum magnitude module, threshold module and decoder have been implemented on Xilinx Spartan-3E devices while the encoder has been implanted on a Virtex-4 device. We have made comparison with Fry et al [8] who have used a Virtex 2000E device to implement a modified fixed order SPIHT scheme and lifting wavelet transform. We have also compared results with Yin-hua Wu et al [19] who have used a Xilinx XC3S5000 to implement the encoder module of a modified SPIHT coder. In Table 7.5, we have given the simulation time for individual modules of our design. Our modules simulate in an order of nanoseconds which is less compared to the other implementations considering our choice of cost effective Spartan 3E device and Virtex-4 devices.

Table 7.5 Comparison Result of SPIHT in terms of Time

Implementation	**Device**	**Frequency**	**Time to completion**
Fry et al [8] Encoder only	Xilinx Virtex2000E	56 MHz	1.101 seconds
Yin-hua Wu et al [19] Encoder only	Xilinx XC3S5000	50 MHz	2.29 ms
Our Maximum Magnitude module only	**Xilinx Spartan 3E XC3S500E**	**74 MHz**	13.581ns
Our Threshold only	**Xilinx Spartan 3E XC3S100E**	**67 MHz**	15.020ns
Our Encoder Module only	**Xilinx Virtex-4 XC4VLX80**	**98 MHz**	318.212 ns
Our Decoder Module only	**Xilinx Spartan 3E XC3S100E**	**69 MHz**	14.489ns

7.4 Throughput and Resource Utilization Comparison

First we shall compare our results with those of Fry et al [8]. Our maximum magnitude module is implemented on a XC3S500E and occupies 3 % of the available resources in slices while giving a throughput of 2368 Mbps. Fry et al [8] have used a Virtex2000E device which has 97% greater number of slices compared to our device and 80% more logic gates compared to the XC3S500E which has a mere 960 slices and is much more cheaper than the Virtex 2000E. It is evident that our maximum magnitude module is far more efficient in terms of area and cost.

Now if we compare our design modules with that of Yin-hua Wu et al [19], each of our modules give a higher throughput in Mbps, in spite of the fact that we have used smaller less costly devices. Corsonello et al [21] have used 15% of total available slices d 18 Block RAMs to implement their encoder module only. In contrast, our encoder may have higher slice utilization but we have not used any Block RAM in our design. In Table 7.6 we have depicted the above discussed comparisons.

Table 7.6 Comparison Result of SPIHT in terms Throughput and Area

Implementation	**Module**	**Device**	**Clock Rate (MHz)**	**Throughput (MPixels/sec)**	**FPGA Area (Slices)**
Fry et al [8]	Maximum Magnitude	Xilinx Virtex2000E	73	146	62%
Fry et al [8]	Encoder and Decoder modules	Xilinx Virtex2000E	56	224	34%
Yin-hua Wu et al [19]	SPIHT	Xilinx XC3S5000	50	200	47%
Jyotheswar et al [20]	SPIHT	Xilinx Virtex4 XC4VLX25	35	280 Mbps	65%
Corsonello et al [21]	Encoder module only	Xilinx XC2V1000	100	-	15 %
Our Design	**Maximum Magnitude only**	**Xilinx Spartan 3E XC3S500E**	**74**	148	**3%**
Our Design	**Threshold module only**	**Xilinx Spartan 3E XC3S100E**	**67**	268	**7%**
Our Design	**Encoder Module only**	**Xilinx Virtex-4 XC4VLX80**	**98**	294	**59%**
Our Design	**Decoder Module only**	**Xilinx Spartan 3E XC3S100E**	**69**	276	**6%**

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This chapter includes the conclusions that we have reached upon at the completion of our work. Also included are suggestions for future improvements in our work and our publications.

8.1 CONCLUSIONS

The SPIHT compression scheme was introduced in 1996 and has since then gone through a number of evolutions. However, to our knowledge through extensive study, we have seen implementations either entirely on hardware or entirely on software.

Our target was to combine the capabilities of hardware and software to implement the original SPIHT. We have implemented the DWT and IDWT on MATLAB which has a large number of built-in functions for easy arithmetic calculations. Our coding and decoding portion was carried out on hardware (FPGA) which provides immense flexibility for blocks with extensive mathematical calculations.

We have successfully managed to implement the SPIHT coder in a time effective architecture while using an average of less that 50% of resources of FPGA devices which are far more cost effective as compared to the devices used by previous implementations. There are three main factors to be considered in design comparison which are cost, area and throughput. There is a trade-off between area and throughput in our design, and we have tried to the give the maximum manageable throughput from a cost effective device at a compromise of area in our encoder module.

8.2 FUTURE WORK

Our recommendations for future work are as follows:

1. Our proposed design architecture can be applied to any square image of dimensions 32x32, 128x 128, 256 x 256 and so on. However for an image with unequal dimensions certain changes must be made in our encoder and decoder architecture.
2. We have implemented our design on cost effective devices with fewer resources compared to high density modern Xilinx FPGA devices which incorporate new features such as DSP Slices and embedded microprocessor. If our design is implemented using embedded features, it can further increase the throughput and simulation time.

8.3 PUBLICATION

Ursila Khan, Arshad Aziz and Valiuddin Abbas, "An Efficient Implementation of SPIHT Algorithm on a Reconfigurable Hardware", IEEE, *8th International Bhurban Conference on Applied Sciences and Technology Conference (IBCAST)*, Islamabad, 11-14 January 2011.

REFERENCES

[1] A. Said, W. A. Pearlman, “A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, pp 243 - 250, June 1996

[2] Subhasis Saha-“Image Compression - From DCT to Wavelets: A Review”, Crossroads, Volume 6, Issue (March2000) Pages: 12 - 21 Year of Publication: 2000, ISSN: 1528-4972

[3] Stephen Wolfram, A New Kind of Science, Notes for Chapter 10: Processes of Perception and Analysis, Section: Data Compression, Page 1069

[4] Satish Kumar, An Introduction to Image Compression, 2001-2003, available at http://www.debugmode.com/imagecmp

[5] ‘The Ear as a Communication Receiver’, English translation of Das Ohr also Nachrichtenempfänger by Eberhard Zwicker and Richard Feldtkeller, Translated from German by Hannes Müsch, Søren Buus, and Mary Florentine. Originally published in 1967; Translation published in 1999.

[6] Charles K. Chui, ‘*An Introduction to Wavelets*, (1992), Academic Press, San Diego, ISBN 0585470901

[7] Stéphane Mallat: A Wavelet Tour of Signal Processing (ISBN 0-12-466606-X)

[8] Thomas W. Fry and Scott Hauck, “SPIHT Image Compression on FPGAs”, Circuits and Systems for Video Technology, IEEE Transactions on , Vol 15 Issue: 9, pp 1138 – 1147, Sept. 2005

[9] Clive Maxfield, book, "The Design Warrior's Guide to FPGAs".Published by Elsevier, 2004. ISBN 0750676043, 9780750676045. Retrieved February 5, 2009

[10] David Manners, Xilinx CEO: FPGA industry at a turning point, available at http://www.electronicsweekly.com, Monday 31 March 2008 18:00, http://www.electronicsweekly.com/Articles/2008/03/31/43433/xilinx-ceo-fpga-industry-at-a-turning-point.htm

[11] Xilinx Inc, “Spartan-3E FPGA Family: Complete datasheet”, *http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf*, April 2008

[12] Xilinx Inc, “Virtex-4 FPGA Family: Complete datasheet”, *http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf*, April 2008

[13] J.M. Shapiro ,1993,”Embedded Image Coding Using Zerotrees of Wavelet Coefficients”, IEEE Transactions on Signal Processing, vol. 41, no. 12, pp.3445-3462.

[14] Ms.Mansi Kambli ,Ms.Shalini Bhatia, Fingerprint Image Compression, Mansi Kambli et al, International Journal of Engineering Science and Technology, Vol. 2(5), 2010, 919-928

[15] J. Malý, P. Rajmic, DWT-SPIHT Image Codec Implementation, Department of Telecommunications, Brno University of Technology, Brno, Czech Republic.

[16] A. Said, W. A. Pearlman, “SPIHT Image Compression: Properties of the Method”, http://www.cipr.rpi.edu/research/SPIHT/spiht1.html

[17] SPIHT Algorithm available at “http://www.ws.binghamton.edu/fowler/fowler%20persona 1%20page/EE523_files/SPIHT_Charts.pdf”

[18] Jie Liang, Presentation on ‘Multimedia Communications Engineering, available at http://www.ensc.sfu.ca/~jiel/courses/424/slides/11_LTWT_5.pdf

[19] Yin-hua Wu, Yin-hua Wu, Long-xu Jin, Hong-jiang Tao, “An improved fast parallel SPIHT algorithm and its FPGA implementation”, Future Computer and Communication (ICFCC), 2010 2nd International Conference, Issue Date: 21-24 May 2010, pp. V1-191 - V1-195, Print ISBN: 978-1-4244-5821-9

[20] J. Jyotheswar, Sudipta Mahapatra, “Efficient FPGA implementation of DWT and modified SPIHT for lossless image compression”, Journal of Systems Architecture 53 (2007) 369–378

[21] Pasquale Corsonello, Stefania Perri, Paolo Zicari, Giuseppe Cocorullo, “Microprocessor-based FPGA implementation of SPIHT image compression subsystems”, Microprocessors and Microsystems Volume 29, Issue 6, 11 August 2005, Pages 299-305.