To the Graduate Council:

I am submitting herewith a thesis written by Suresh S. Polisetty entitled "Hardware Acceleration of the Embedded Zerotree Wavelet Algorithm." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin
Major Professor

We have read this thesis and
recommend its acceptance:

Gregory D. Peterson

Mohammad Ferdjallah

Accepted for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

HARDWARE ACCELERATION OF THE EMBEDDED ZEROTREE WAVELET ALGORITHM

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Suresh S. Polisetty

December 2004

Dedicated to my family, teachers and friends

ACKNOWLEDGEMENTS

Foremost, I would like to express my deepest gratitude to my advisor, Dr.Bouldin, for his excellent guidance and enormous support during my graduate study at The University of Tennessee, Knoxville.

Many thanks to Dr.Gregory Peterson and Dr. Mohammed Ferdjallah for serving on my thesis committee.

I personally thank Dr. Gregory Peterson for his great motivation and for providing me with financial assistance during the first semester, which were the critical days being in a far-off land.

I am grateful to Dr. A.J. Baker, Director, UT CFD Laboratory and the Radiation Safety Department for assisting me financially during my graduate study.

I am deeply indebted to my parents for the support and the motivation they provided me to explore higher levels of education.

My special thanks to Mrs. Marilyn Schmeichel for her valuable time spent in proofreading and providing valuable suggestions to my documentation. I also thank Dr.Chandra Tan, Sidd, Mahesh, Sampath, for their helpful suggestions. I thank all my buddies especially Seeram for their endless support. Nothing would be possible without their good wishes.

Last, and surely not the least, my heart full thanks to my dearest friend and soon to be wife, Aruna, for her unrelenting support and being there for me.

ABSTRACT

The goal of this project was to gain experience in designing and implementing a microelectronic system to acclerate the execution of a time-consuming software algorithm, the Embedded Zerotree Wavelet (EZW), which is used in multimedia applications. The algorithm was implemented using MATLAB to be certain it was fully understood and to serve as a validation reference. Then, the algorithm was mapped into a hardware description language, VHDL, and its resulting implementation verified with the golden reference. The hardware description was then targeted to a field-programmable gate array (FPGA).

Significant acceleration was achieved since the hardware implementation in a FPGA (Xilinx Virtex-1000E using a 8.315 MHz clock) ran 10,000 times faster than the MATLAB implementation on a SUN-220 workstation. Additional speedup exploiting the parallel capabilities of the FPGA was not achieved since the EZW algorithm utilizes only sequential operations.

TABLE OF CONTENTS

CHAPTER 1 ..... **1**

INTRODUCTION **1**

1.1 GOAL AND APPROACH 1

1.2 ALGORITHM SELECTION 1

CHAPTER 2 ..... **3**

BACKGROUND **3**

2.1 VHDL – DESIGN FLOW 3

2.2 PILCHARD – A RECONFIGURABLE COMPUTING PLATFORM 6

2.3 EMBEDDED CODING 10

2.4 ZEROTREE STRUCTURE 11

2.5 WAVELET DECOMPOSITION: DISCRETE WAVELET TRANSFORM (DWT) 13

2.5.1 DCT Vs DWT *18*

CHAPTER 3 ..... **21**

APPROACHES & IMPLEMENTATION **21**

3.1 CONCEPTS OF EZW 21

3.2 EZW – THE ALGORITHM 23

3.2.1 Dominant Pass *26*

3.2.1 Subordinate Pass *29*

3.3 EZW – AN EXAMPLE 30

3.4 SOFTWARE IMPLEMENTATION 40

3.4.1 EZW specifications *41*

3.4.2 Deviations from EZW specifications *41*

3.4.3 Decoding the Bitstream generated by EZW *44*

3.5 HARDWARE IMPLEMENTATION 46

3.5.1 Explicit Design Flow of the Pilchard RC Platform *46*

3.5.2 Design Details *50*

3.5.2.1 Encoder 52
3.5.2.2 Controller 52
3.5.2.3 Dual Port – RAM 58
3.5.3 Pre-Synthesis Simulations 59
3.5.4 Synthesis 59
3.5.5 Place and Route (PAR) 59
3.5.6 Bit file generation 59
CHAPTER 4 64
RESULTS 64
4.1 SOFTWARE 64
4.1.1 Test Image – Lena 64
4.1.2 Test Image – Barbara 67
4.1.3 Test Image – Goldhill 73
4.2 HARDWARE 79
4.2.1 Area and Speed 85
4.2.2 Limitations 85
4.2.2.1 Limitaion#1 85
4.2.2.2 Limitaion#2 86
4.3 HARDWARE VS. SOFTWARE 86
4.4 SPEEDUP AND DESIRED ARCHITECTURE 87
4.4.1 Speedup Including the Time for Data Transfer 88
4.4.2 Other Possibilities to Achieve Speedup 89
CHAPTER 5 93
CONCLUSIONS AND FUTURE WORK 93
5.1 CONCLUSIONS 93
5.2 FUTURE WORK 93
REFERENCES 94
VITA 98

LIST OF TABLES

Table 1 Xilinx® Virtex™ FPGA Device XCV1000E Product Features [13] 8

Table 2 Features of the Pilchard RC platform [12] 8

Table 3 First Dominant Pass 38

Table 4 First Subordinate Pass 38

Table 5 Arithmetic Coding of Symbols for Dominant Pass 43

Table 6 Arithmetic Coding of Symbols for Subordinate Pass 43

Table 7 Detailed operations of each State of the Encoder 55

Table 8 Detailed operations of each State of the Controller 57

Table 9 Experimental results of Lena 67

Table 10 Experimental results of Barbara 73

Table 11 Experimental results of Goldhill 79

Table 12 Area and Speed 86

Table 13 Comparison between DCT and EZW on ASIC. 92

Table 14 Possible EZW Speed-ups on Different Platforms 92

LIST OF FIGURES

Figure 1 Digital Design Flow 4

Figure 2 The Pilchard Board [12] 7

Figure 3 Block Diagram of Pilchard [12] 9

Figure 4 Parent-Childe Dependencies [1] 12

Figure 5 Scanning Order [1] 14

Figure 6 First Stage of Discrete wavelet Transformation [1] 15

Figure 7 Second-Stage Wavelet Decomposition [1] 15

Figure 8 Original image Lena 512 x 512 16

Figure 9 One-Scale DWT Decomposition 17

Figure 10 Two scale-Dimensional DWT Decomposition 19

Figure 11 DCT Vs DWT [14] 20

Figure 12 Typical Flows of Data of Image Encoder 21

Figure 13 Flow-chart of EZW Encoder 25

Figure 14 Flow-chart of Dominant Pass 27

Figure 15 Flow-chart of Subordinate Pass 31

Figure 16 An 8x8 Sample Image 31

Figure 17 Step#1 of Dominant Pass Figure 18 Step#2 of Dominant Pass 33

Figure 19 Step#3 of Dominant Pass Figure 20 Step#4 of Dominant Pass 33

Figure 21 Step#5 of Dominant Pass 34

Figure 22 Step#6 of Dominant Pass 35

Figure 23 Step#7 of Dominant Pass 35

Figure 24 Step#8 of Dominant Pass....37
Figure 25 Step#9 of Dominant Pass....37
Figure 26 Morton-Scan Order....42
Figure 27 Flow-chart for the Decoder....45
Figure 28 Block Diagram of the EZW Implementation....46
Figure 29 Explicit design Flow for Pilchard RC (Courtesy: Dr. Chandra Tan)....48
Figure 30 Flow of Design in the PAR tools....49
Figure 31 Hierarchy of the modules....51
Figure 32 A sample FSM....53
Figure 33 VHDL Representation of a sample FSM....53
Figure 34 FSM of the Encoder....54
Figure 35 FSM of the Controller....56
Figure 36 Read/Write Operations of Xilinx Dual-Port RAM [17]....58
Figure 37 Waveform indicating inputs of encoder module....60
Figure 38 Waveform showing done signal becoming 1 at 49558 ns....60
Figure 39 Waveform showing parith and pcore signals, and start becoming 1....61
Figure 40 Waveform indicating done signal becoming 1 at 317400....62
Figure 41 Layout of the Design using Synplify- Pro....63
Figure 42 Test Image – Lena 512 x 512 Original....65
Figure 43 Lena 512 x 512, 6-scale DWT Decomposition....66
Figure 44 L-Compr. Ratio 1024 : 1....68
Figure 45 L-Compr. Ratio 512 : 1....68
Figure 46 L-Compr. Ratio 256 : 1....68

Figure 47 L-Compr. Ratio 128 : 1 68
Figure 48 L-Compr. Ratio 64 : 1 69
Figure 49 L-Compr. Ratio 32 : 1 69
Figure 50 L-Compr. Ratio 16 : 1 69
Figure 51 L-Compr. Ratio 8 : 1 69
Figure 52 Reconstructed Lena Image using all 42848 bytes 70
Figure 53 Test Image – Lena 512 x 512 Original 71
Figure 54 Barbara 512 x 512, 6-scale DWT Decomposition 72
Figure 55 B-Compr. Ratio 1024 : 1 74
Figure 56 B-Compr. Ratio 512 : 1 74
Figure 57 B-Compr. Ratio 256 : 1 74
Figure 58 B-Compr. Ratio 128 : 1 74
Figure 59 B-Compr. Ratio 64 : 1 75
Figure 60 B-Compr. Ratio 32 : 1 75
Figure 61 B-Compr. Ratio 16 : 1 75
Figure 62 B-Compr. Ratio 8 : 1 75
Figure 63 Reconstructed Barbara Image using all 45504 bytes 76
Figure 64 Test Image – Goldhill 512 x 512 Original 77
Figure 65 Goldhill 512 x 512, 6-scale DWT Decomposition 78
Figure 66 G-Compr. Ratio 1024 : 1 80
Figure 67 G-Compr. Ratio 512 : 1 80
Figure 68 G-Compr. Ratio 256 : 1 80
Figure 69 G-Compr. Ratio 128 : 1 80

Figure 70 G-Compr. Ratio 64 : 1 81

Figure 71 G-Compr. Ratio 32 : 1 81

Figure 72 G-Compr. Ratio 16 : 1 81

Figure 73 G-Compr. Ratio 8 : 1 81

Figure 74 Reconstructed Goldhill Image using all 45428 bytes 82

Figure 75 MATLAB Implementation for Multiple Compression Ratios 83

Figure 76 Verification of the Hardware Implementation 84

Figure 77 An example system-on-chip platform [26] 91

CHAPTER 1
INTRODUCTION

1.1 Goal and Approach

The goal of this project was to gain experience in designing and implementing a microelectronic system to acclerate the execution of a time-consuming software algorithm used in numerous applications. The project began with the selection of a candidate algorithm and its implementation using MATLAB to be certain it was fully understood and to serve as a validation reference. Then, the algorithm was mapped into a hardware description language, VHDL, and its resulting implementation verified with the golden reference. The hardware description could then be targeted to either a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC).

1.2 Algorithm Selection

Many evolving multimedia applications require transmission of high quality images over the network, which in turn need efficient image coding methods to meet challenges such as coding efficiency, scalability, target compression rates, low delay, low power consumption and implementation simplicity. Image processing is normally done using different software packages like PhotoShop and MATLAB. These software applications execute on a central processing unit (CPU) of a computer, which executes image manipulation routines sequentially. Because the CPU must be shared with other applications and is not able to execute the image manipulations in parallel,

performance suffers. The research described in this thesis involves implementing image-processing functions in a FPGA, which serves as a CPU coprocessor to speed up processing times. This can be accomplished by taking advantage of pipelining and/or parallel processing implemented on dedicated hardware. Such extreme parallelism is almost impossible with traditional CPU architectures.

A decade ago, a group of efficient image coders based on wavelet hierarchical decomposition was developed and resulted as one of the most promising techniques to meet the aforementioned challenges for image coding. The idea of grouping wavelet coefficients at different scales and predicting zero coefficients across scales was introduced. In [1], Shapiro proposed an Embedded Zerotree Wavelet (EZW) coding scheme that not only has provided excellent coding performance, but also has a fully embedded bit stream. The EZW algorithm is a simple, easy to implement, and an effective image compression technique. The EZW algorithm uses the concepts of Discrete Wavelet Transform, Embedded Coding, Zerotree Coding and loss-less Arithmetic Coding.

CHAPTER 2
BACKGROUND

2.1 VHDL – Design Flow

Hardware can be described by programming languages like VHDL, which stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL modules can be simulated to test the functional behavior of the hardware implementations of the design as well as its timing constraints. VHDL descriptions can be "synthesizable" which means the behavioral description can be translated into physically realizable circuits, such as NAND gates, XOR gates and Flip-flops, using CAD (Computer Aided Design) tools. Synthesis can be targeted to programmable logic devices, such as FPGAs or to application-specific integrated circuits (ASICs). FPGAs are commonly designed to be reprogrammable, so they are often used to test algorithms. ASICs are chips that are designed with a specific purpose in mind, and are generally not reprogrammable, but they are usually faster than FPGAs. However, FPGAs available in recent days are made of circuits with millions of transistors, and are extensively used for the applications like prototyping and high performance reconfigurable computing.

Figure 1 shows the basic flow for the design of digital circuits in ASICs/FPGAs. The problem statement specifies the requirements of the design. Then code is generated to meet the specifications. Any hardware description language, for example VHDL, is used for the coding. Now the code is simulated to achieve the correct functionality. If the code needs any changes, then it is repaired, and again

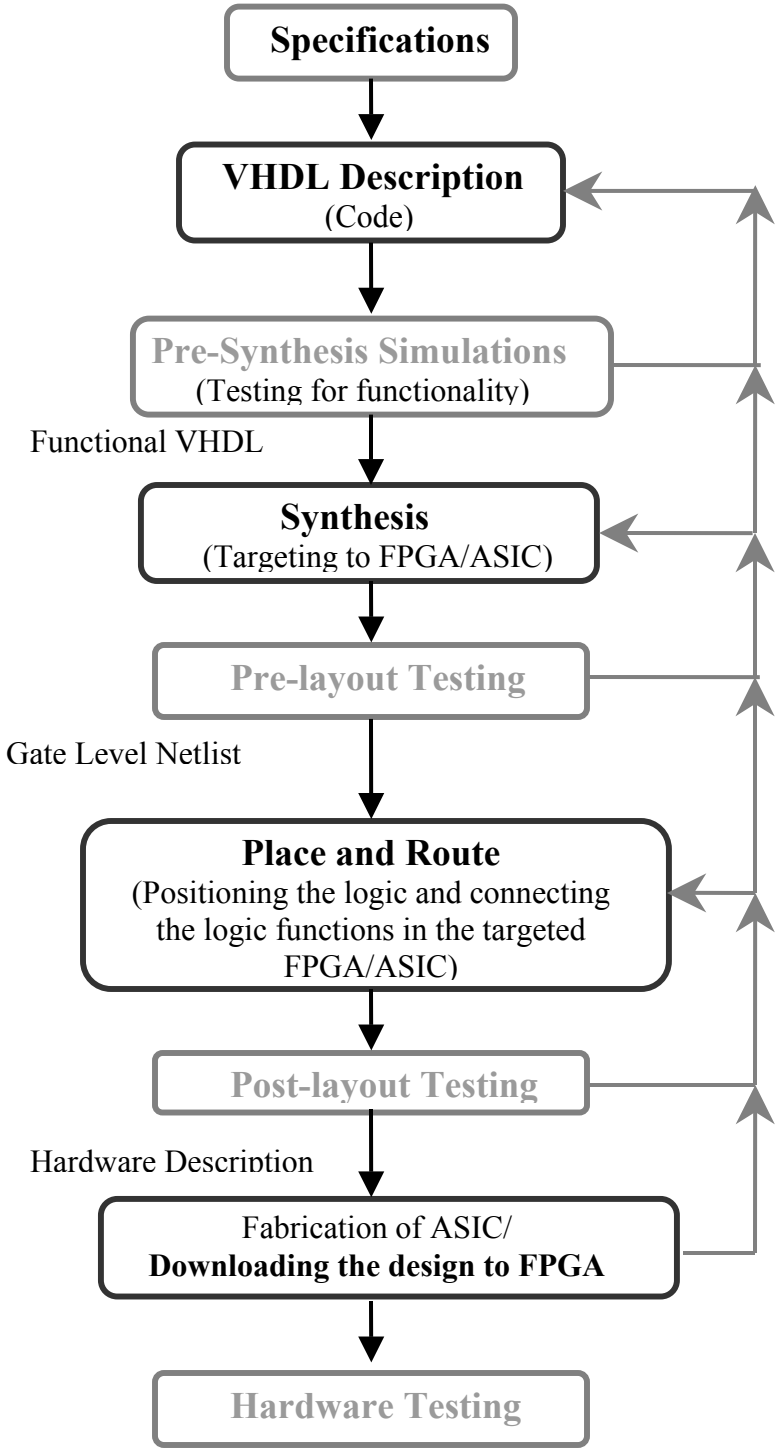


Figure 1 Digital Design Flow

simulated. This process continues till the required functionality is achieved. Once the code is functional, then it is synthesized using synthesizing CAD tools, which can take care of priority design constraints like delay, area and power. There is always a tug-of-war between performance (area and delay) and power-consumption. For example, think about the demand for a "Notebook PC (laptop)", which can easily fit in a backpack. It is desirable that the laptop weight and power be reduced in half yet run twice as fast and cost the same as an existing model.

Coming to the point, synthesis is an automatic method of converting a higher-level abstraction, such as a behavioral description, to a lower level abstraction, such as a gate-level netlist. But, all the VHDL statements are not synthesizable. Another VHDL module apart from the actual design modules, known as the "test-bench", is used in the simulations. The port structure, which specifies the input-output pins, of the test-bench is exactly opposite to the port structure of the top module of the design. The purpose of a test-bench is limited to provide test vectors to the input pins, and to receive the responses from the output pins. A test-bench cannot be synthesized.

Once the design is synthesized and the timing constraints are met, then the design is ready for the next step, i.e., Place and Route (PAR). PAR can be defined as the process of mapping a synthesized netlist in terms of physical location (place) and the interconnection of the corresponding blocks (route) [11]. At this stage the exact timing constraints of the design will be revealed and the area of the design can be a accurately measured. If there are any failed constraints, then the design has to be modified from the beginning. In the case of a FPGA, if the PAR is successful, an **.ncd** (Native Circuit Description) file is generated, which will be used to create a layout and

configuration or **.bit** file. In case of an ASIC, a GDSII file is generated, which can be used in the fabrication process.

2.2 Pilchard – A Reconfigurable Computing Platform

The Pilchard Reconfigurable Computing (RC) board (Figure 2) developed at the Chinese University of Hong Kong [12] is the hardware system used for the implementation of this design. This RC board accommodates a million-gate FPGA, the Xilinx® Virtex™1000E (XCV1000E). The product features of the XCV1000E, as obtained from the manufacturer's website, are listed in the Table 1. The Pilchard uses 133MHz synchronous dynamic RAM Dual In-line Memory Modules (DIMMs) interfacing with the CPU. Compared to the usual commercially available RC boards with traditional Peripheral Component Interconnect (PCI) interface, the DIMM interface offers higher bandwidth for communication between the host processor and the RC board and lower latency, yet, is easier to interface.

The block diagram of the Pilchard board is shown in the Figure 3. The Table 2 gives the features of the board. This advantage of the communication between the host processor and the RC board is accomplished by a software interface program executing on the host processor. There are four API functions of the software interface:

I. void read64(int64, char *) - To read 64 bits from Pilchard

II. void write64(int64, char *) - To write 64 bits to Pilchard

III. void read32(int, char *) - To read 32 bits from Pilchard

IV. void write32(int, char *) - To write 32 bits from Pilchard

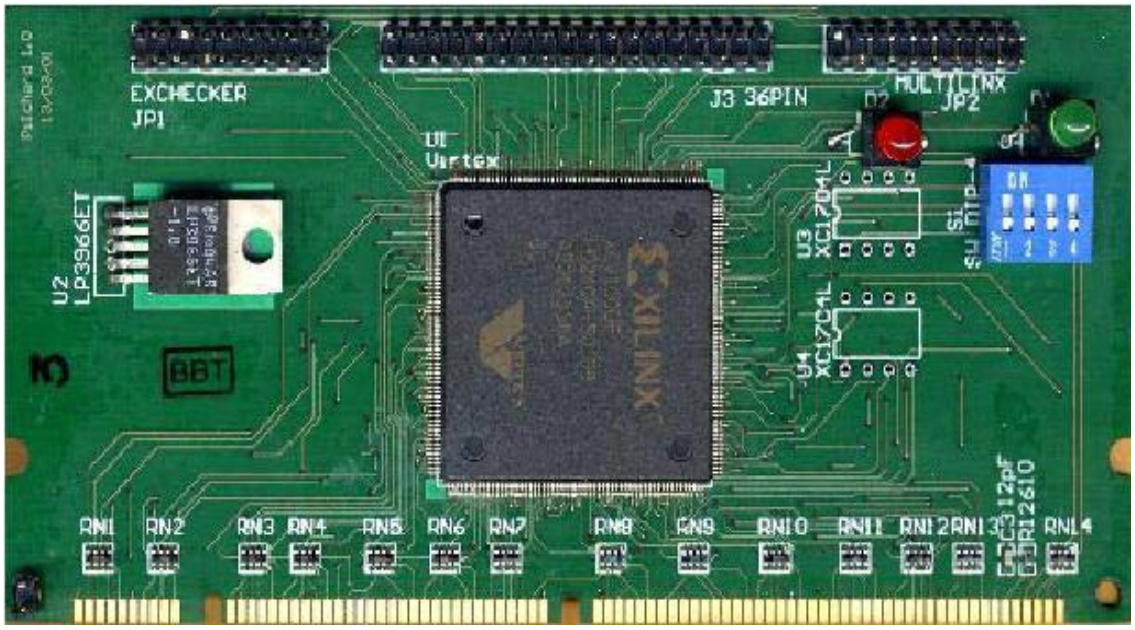


Figure 2 The Pilchard Board [12]

Table 1 Xilinx® Virtex™ FPGA Device XCV1000E Product Features [13]

Feature	Specification
Package used in Pilchard	HQ240 (32mm × 32mm)
CLB Array (Row × Col.)	64×96
Logic Cells	27,648
System Gates	1,569,178
Max. Block RAM Bits	393,216
Max. Distributed RAM Bits	393,216
Delay Locked Loops (DLLs)	8
I/O Standards supported	20
Speed Grades	6,7,8
Available User I/O	158 pins (for package PQ240) max.660

Table 2 Features of the Pilchard RC platform [12]

Feature	Specification
Host Interface	DIMM Interface 64-bit Data I/O 12-bit Address Bus
External (Debug) Interface	27 – Bits I/O
Configuration Interface	X-checker, MultiLink and JTAG
Maximum System Clock Rate	133 MHz
Maximum External Clock Rate	240 MHz
FPGA Device	XCVE1000E-HQ240-6
Dimension	133mm × 65mm × 1mm
OS Supported	GNU/LINUX
Configuration Time	16s Using Linux download program

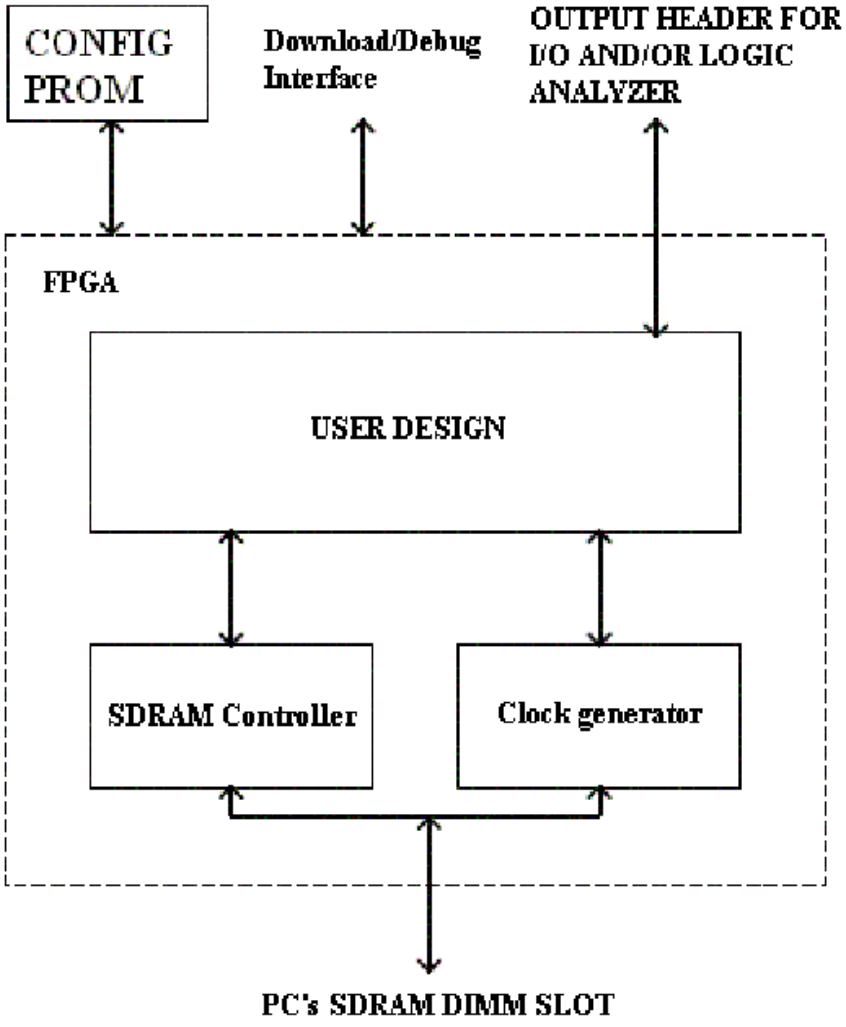


Figure 3 Block Diagram of Pilchard [12]

“int64” is a data type provided by “iflib.h” as a two-element integer array. The FPGA is configured with the design bit-stream by " download.c.”

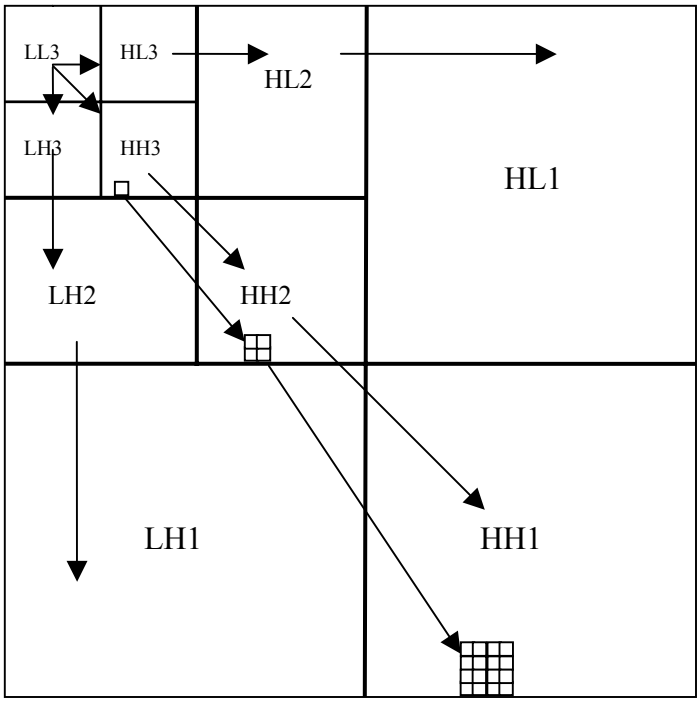
2.3 Embedded Coding

An “Embedded Coding” can be defined as representing a sequence of binary decisions that distinguish an image from the “null”, or all gray, image. During the image encoding, all lower frequency codes are “embedded” at the beginning of the bit stream, and the bits are arranged in order of importance. An encoder following the embedded coding technique can terminate the encoding at any point thereby allowing a target bit rate to be met exactly. Some target parameters can be monitored to stop the encoding exactly when the target rate is met. A decoder is also capable of stopping decoding at any point and can reconstruct the image corresponding to all lower frequency encoding.

Binary finite precision representation of real numbers is a proper example of embedded coding. Binary representation of all real numbers is possible using a string of binary digits. If a bit is added to the right of a floating-point binary string, the precision of the decimal equivalent of the binary string will be increased. It is obvious that, the addition of the bits can cease at any time and provide the “best” representation of the real number achievable within the framework of the binary digit representation. Similarly, the embedded coder can cease at any time and can provide the “best” representation of an image achievable within its frame work **[1]**.

2.4 Zerotree Structure

An image-based data structure arranged in parent-child order, called simply a *tree*, is a set of wavelet coefficients corresponding to the same spatial location and orientation. The zerotree-based image coders are based on the assumption that if there are insignificant coefficients in low frequency subbands in a tree, then the probability of corresponding coefficients in the higher frequency subbands being insignificant is higher. If all the coefficients of a tree are insignificant with respect to a given threshold, then the tree is called a *zerotree*. When encoding an image, fewer bits are sufficient to represent the zerotree, whereas the non-zerotree structures require substantial number of bits **[8]**. In the zerotree-based system, every coefficient at any lower frequency subband has a relationship with a group of coefficients at the next higher frequency subband at the same spatial location, except the highest frequency subbands. The coefficient at the lower frequency subband is called the *parent*, and all coefficients corresponding to the same spatial location at the next higher frequency subband of analogous orientation are called *children.* Also, the set of all the coefficients corresponding to the same spatial location, relating to a parent, at the subsequent higher frequency subbands are called descendants. Similarly, the set of all the coefficients corresponding to the same spatial location, relating to a child, at the subsequent lower frequency subbands are called ancestors. The parent-child dependencies are shown in Figure 4 **[1]**. The coefficients are scanned in a particular order, which assures the fact that all the parent nodes must be scanned before their children. For a 2-scale wavelet transform, the scanning of the coefficients begins with LL_2, the lowest frequency band, and follows to the HL_2, LH_2 and HH_2. After finishing the coarser scale, it moves to the



(Note) Parent-Child dependencies of subbands: Note that the arrow points from the subband of the parents to the subband of the children. The lowest frequency subband is the top left, and the highest frequency subband is at the bottom right. Also shown is a wavelet tree consisting of all of the dependencies of a single coefficient in subband HH3. The coefficient in HH3 is a zerotree root if it is insignificant and all of its descendents are insignificant.

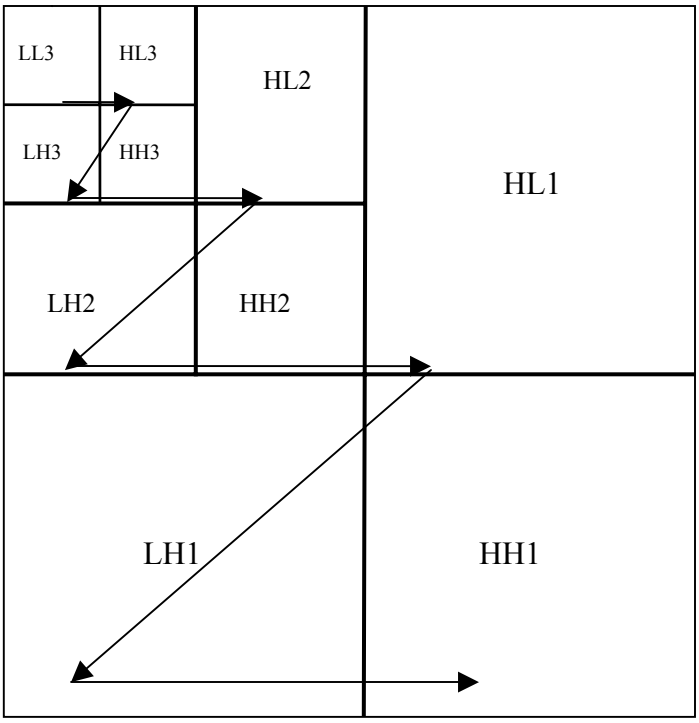
Figure 4 Parent-Childe Dependencies [1]

next finer scale, which is the 1-scale in this case. So, the order of scanning moves from HH_2 to HL_1, HL_1 to LH_1 and finally ends at the highest frequency subband HH_1. The scanning order is shown in Figure 5 [1].

2.5 Wavelet Decomposition: Discrete Wavelet Transform (DWT)

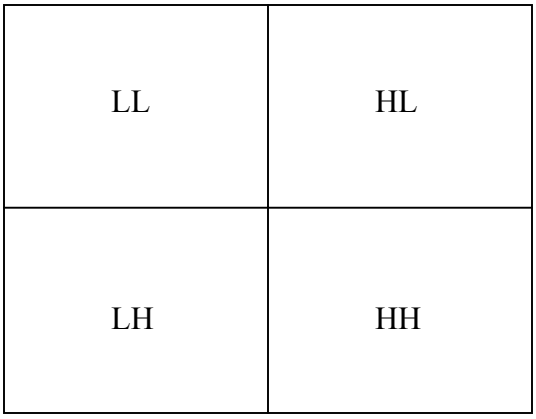
The discrete wavelet transform used in the EZW algorithm, proposed by Shapiro, is similar to a hierarchical subband system, in which the subbands are logarithmically spaced in frequency and represent octave-band decomposition. The original image is split into subbands and sub-sampled as shown in Figure 6 [1]. Each coefficient represents a spatial area corresponding to approximately ¼ of the area of the original image. The four subbands are formed from the vertical and horizontal filtering process. The subbands labeled LH_1, HL_1 and HH_1 represent the higher frequency wavelet coefficients. The subband LL_1 is decomposed again and critically sampled as shown in Figure 7 [1], to obtain the next coarser scale of wavelet coefficients. This process is recurrent until the target scale is reached. The coarser the scale, the larger will be the representation of the spatial area of the coefficients of images, but the frequencies are narrower. There are three subbands at each scale. The fourth and lowest frequency subband is located at the left top and contains the information of all coarser scales [1].

Here is an example of the original image "Lena" of size 512x512 shown in Figure 8, undergoing dyadic decomposition into subbands. The image shown in Figure 9 is after a one- scale DWT Decomposition. It can be noticed that the image in the subband LL1 is better than the image in HL1, so that HL1's is better than LH1's



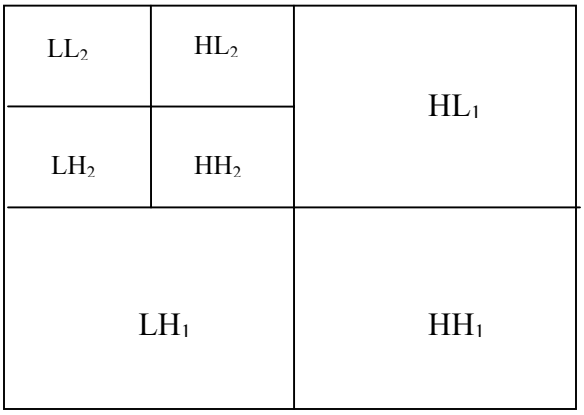
(Note) Scanning Order: Scanning order of the subbands for encoding a significance map: Note that parent must be scanned before children. Also note that all positions in a given subband are coded before moving to he next. in a given subband are scanned before the scan moves to the next subband

Figure 5 Scanning Order [1]



(Note) First stage of discrete wavelet transformation: The image is divided into four subbands using separable filters. Each coefficient represents a spatial area corresponding to approximately a 2x2 area of the original picture. The low frequencies represent a bandwidth approximately corresponding to $0< |\omega | < \pi/2$, whereas the high frequencies represent the band from to $\pi/2 < |\omega | < \pi$. The four subbands arise from separable application of vertical and horizontal filters.

Figure 6 First Stage of Discrete wavelet Transformation [1]



(Note) Two-scale wavelet decomposition: The image is divided into four subbands using separable filters. Each coefficient in the subbands LL2, LH2, HL2 and HH2 represents a spatial area corresponding to approximately a 4x4 area of the original picture. The low frequencies at this scale represent a bandwidth approximately corresponding to $0< |\omega | < \pi/4$, whereas the high frequencies represent the band from to $\pi/4 < |\omega | < \pi/2$.

Figure 7 Second-Stage Wavelet Decomposition [1]

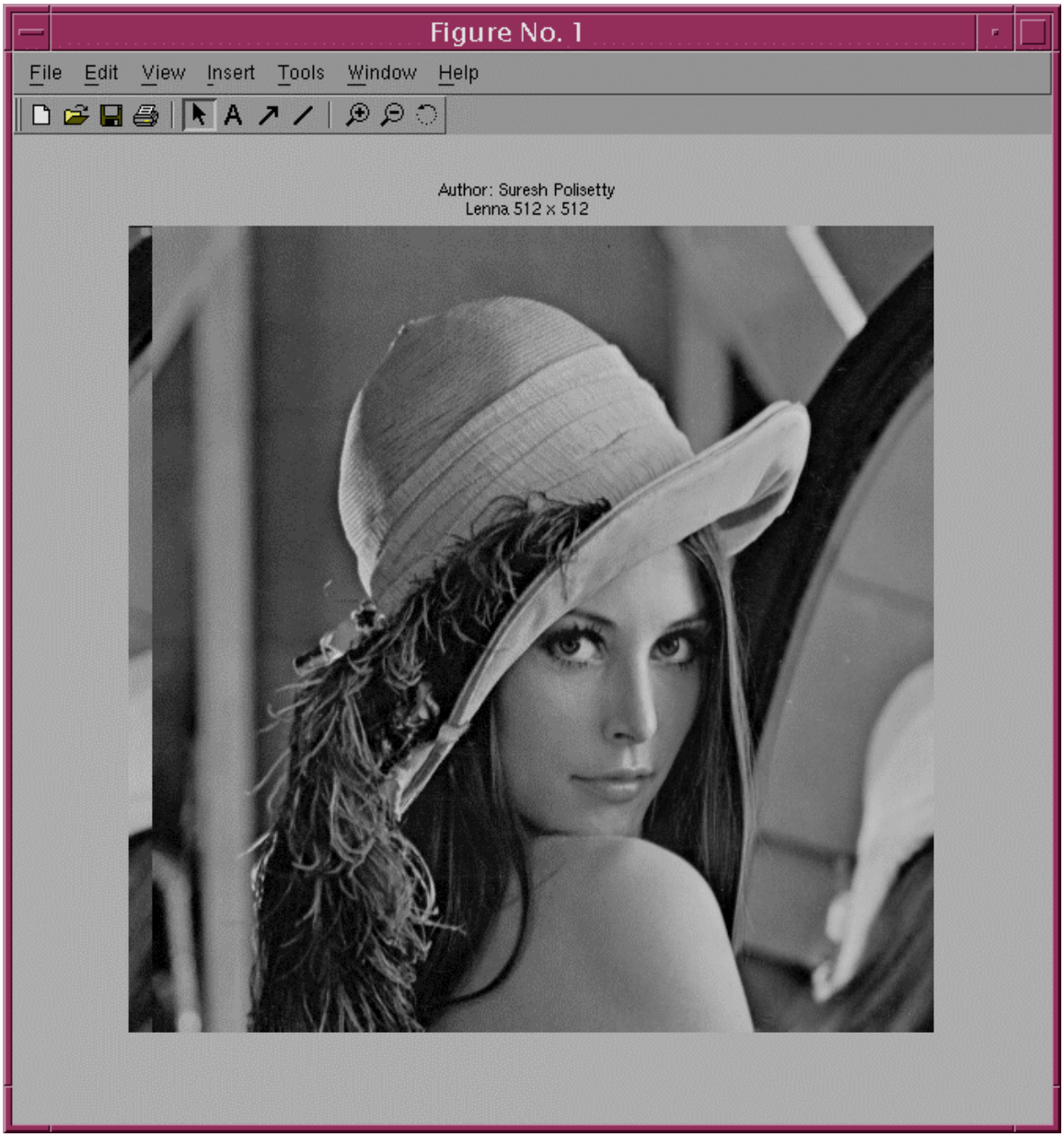


Figure 8 Original image Lena 512 x 512

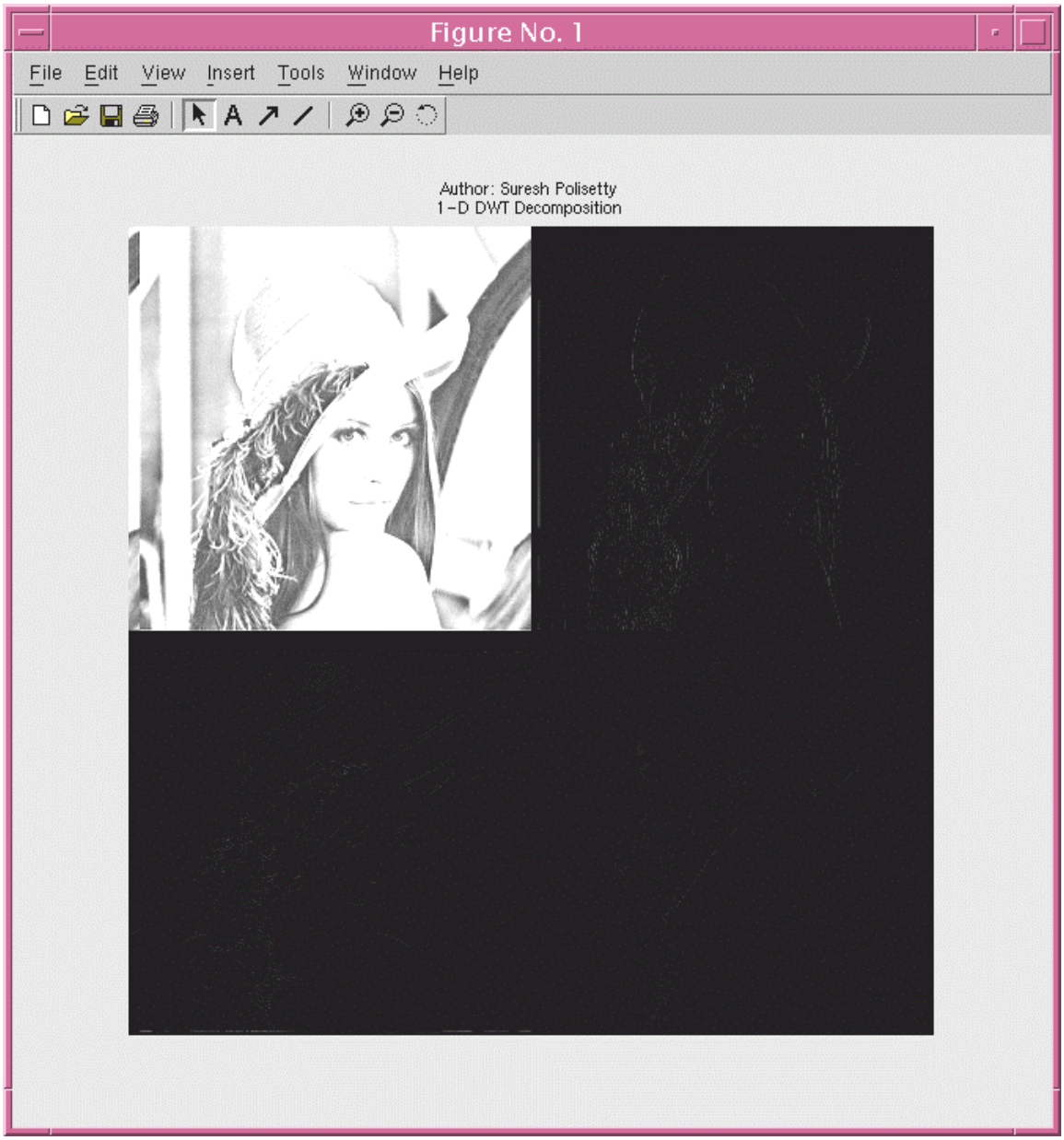


Figure 9 One-Scale DWT Decomposition

and LH1's is better than HH1,which is due to the fact that the most significant information of the image will be stored in the lowest frequency subband. The least significant information of the image is stored in the highest frequency subband and the order of significance follows the same order as the order shown previously in Figure 5 [1]. The two-scale DWT decomposition into subbands is shown in Figure 10.

2.5.1 DCT Vs DWT

The Discrete Cosine Transform (DCT) is the traditional transformation method used in image compression techniques such as Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MPEG1 & MPEG2). Developed in the early 1990's, the Discrete Wavelet Transform (DWT) has gained popularity over the DCT. The latest compression techniques like JPEG2000 and MPEG4 use DWT. Unlike the DCT, coefficients from the DWT are stable under the presence of discontinuities in the signal to be coded. The DWT only requires a piecewise smooth signal, where as the DCT requires a globally smooth signal. Most video and image compression implemented using the Discrete Wavelet Transform does not exhibit the blocking, also known as tiling, artifacts seen with the block Discrete Cosine Transform. DWT-based image compression often outperforms block DCT compression if evaluated using the Peak Signal to Noise Ratio (PSNR) or Mean Squared Error (MSE) metric (these are mathematically equivalent). The subjective quality of images compressed with the DWT can appear better than block DCT methods for the same compression ratio. Figure 11 [14] shows the comparison between JPEG and JPEG2000 compressed images.

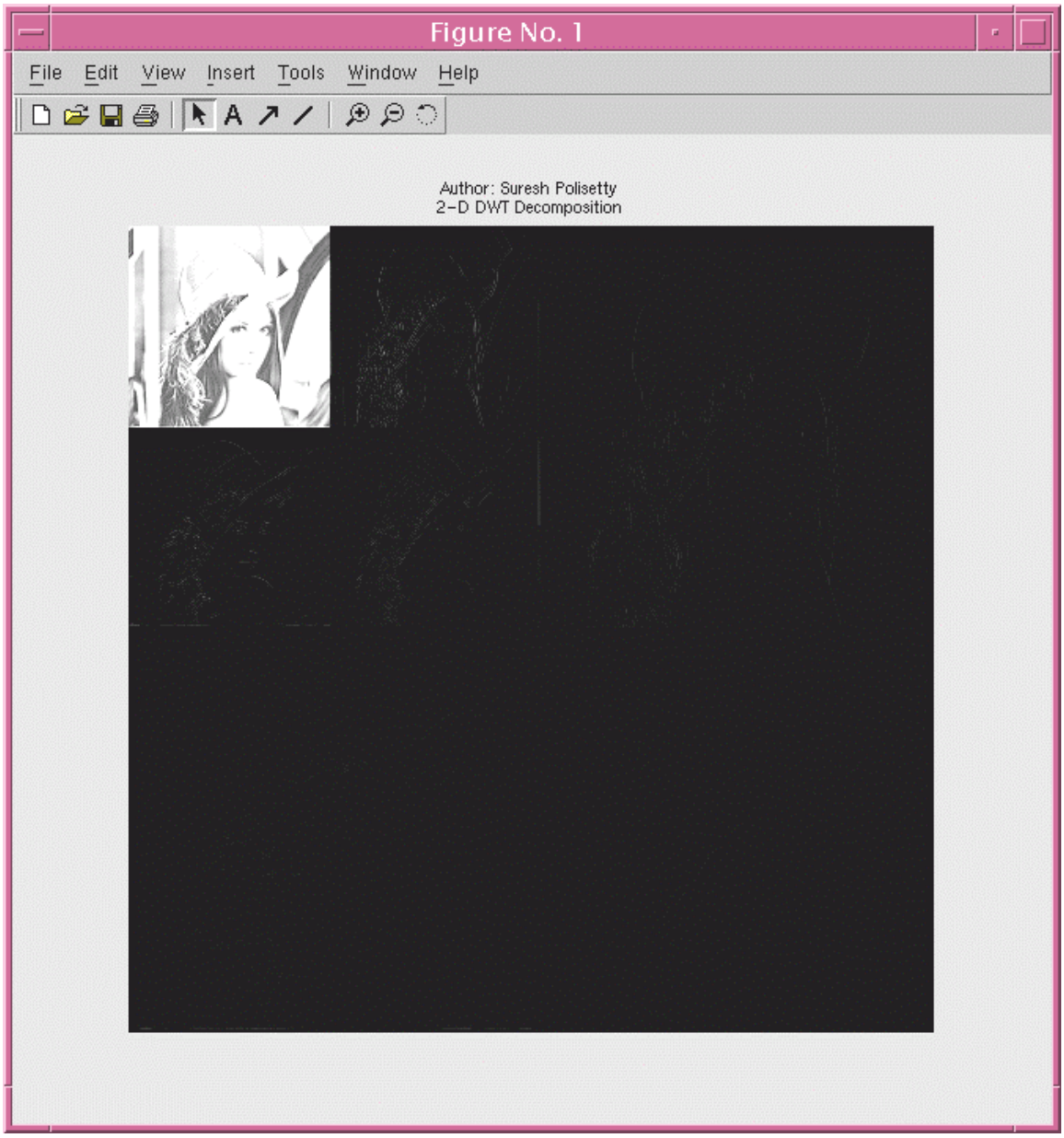


Figure 10 Two scale-Dimensional DWT Decomposition

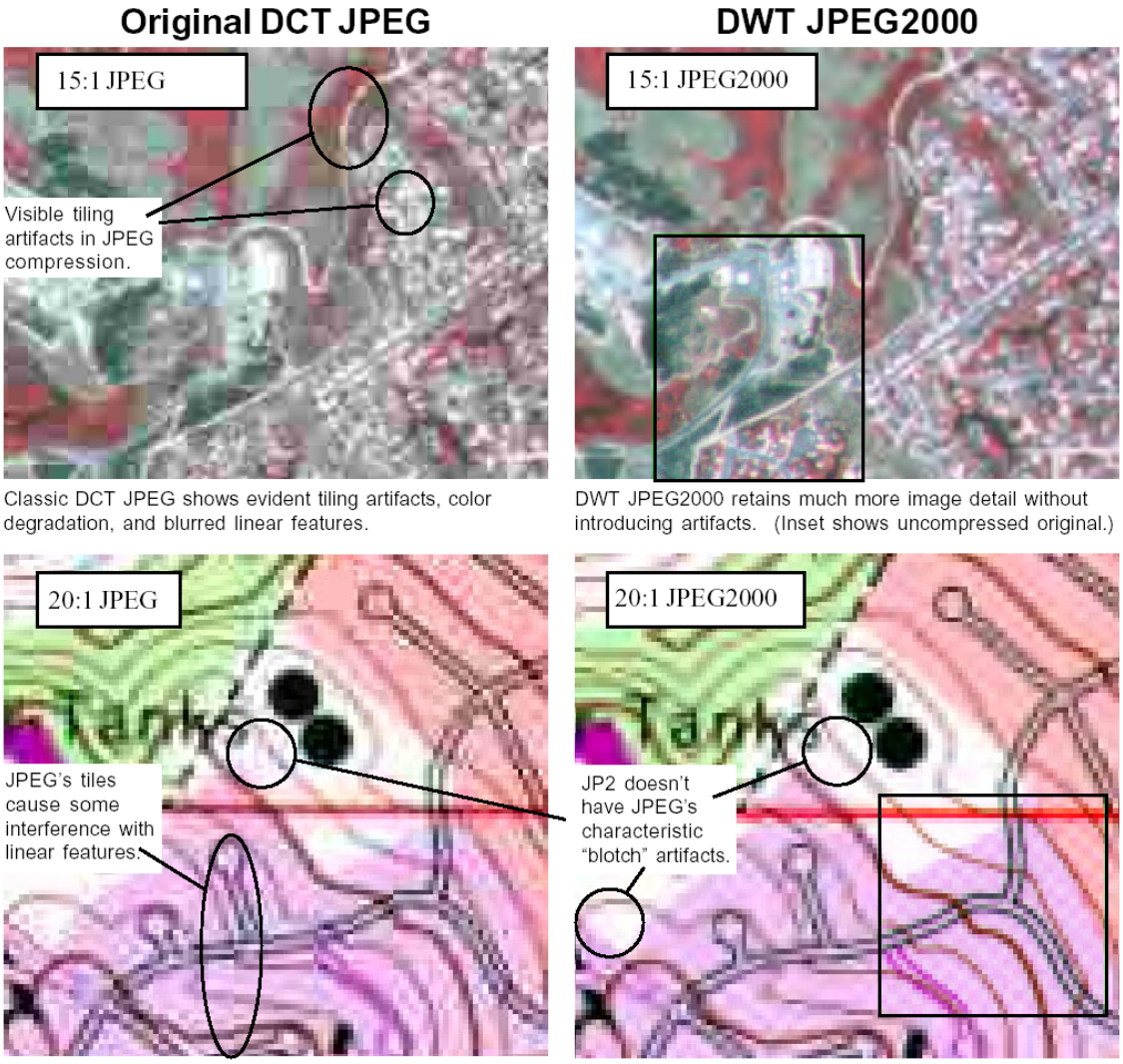


Figure 11 DCT Vs DWT [14]

CHAPTER 3
APPROACHES & IMPLEMENTATION

3.1 Concepts of EZW

The EZW image encoder follows the typical flow of data as shown in the Figure 12, and has three basic steps: 1) Transformation, 2) Quantization and 3) Compression.

(Step 1) Transformation. EZW uses the Discrete Wavelet Transform (DWT) to transform the original image. In order to perform the DWT, the image has to be a square image, and its row/column size must be an integer power of 2. So, technically, the EZW is applicable to the square images of sizes in integer powers of 2 (for example, image sizes like 128 x 128 or 512 x 512).

This transformation is theoretically lossless, although this may not always be the case. The purpose of the transformation is to generate decorrelated coefficients, which means it removes all the dependencies between samples.

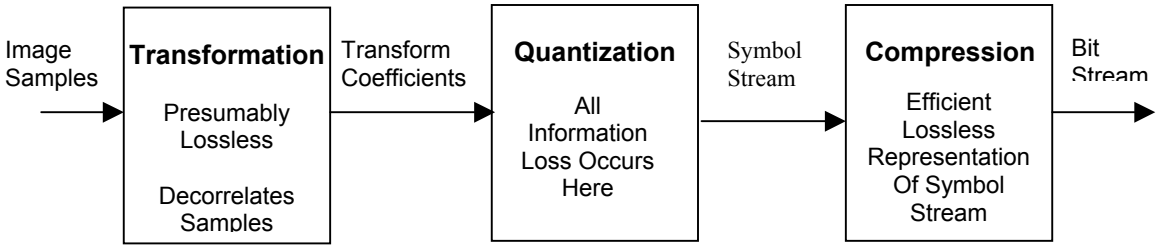


Figure 12 Typical Flows of Data of Image Encoder

(Step 2) Quantization. This step involves the quantization of transformed coefficients. Thus, the entropy of the resulting distribution of the bin indexes is small enough that the symbols can be entropy coded at some low target bit rate. Quantizers are symmetrically read. Assuming the central index is zero, which treats positive or negative indexes alike, all quantizers are set to be symmetric. The main advantage of symmetry is that it saves the bits needed to represent the symbols since encoding of a non-zero coefficient requires at least one bit per sign. An entropy code can be designed using the probabilities of the bin indices as the fraction of coefficients in which the only absolute values of bin indexes are involved. Entropy of the symbols H can be expressed as

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p) + (1-p) [1 + H_{NZ}],$$

where p = probability that a transform coefficient is quantized to zero, and

H_{NZ} = conditional entropy of the absolute values of the quantized coefficients conditioned on them being non-zero.

The EZW uses Successive Approximation Quantization (SAQ). SAQ is chosen to achieve a multiprecision representation of the coefficients and to facilitate the embedded coding. The significance of the wavelet coefficients with respect to a monotonically decreasing series of thresholds, T_i, is determined by using SAQ. For each threshold, T_i, the positions of the significant and the insignificant coefficients are indicated in significance maps.

(Step 3) Compression. The concept of a zerotree data structure is applied in the compression process of the significance map. Each wavelet coefficient is compared with the threshold, T_i, to determine its significance. In addition to encoding

the significance map, further encoding of significant coefficients is done using signs. All the significant coefficients are encoded into only four signs: 1) zerotree root, 2) isolated zero, 3) positive significant, and 4) negative significant. Encoding into symbols makes embedded coding handy. EZW follows adaptive arithmetic coding for compression. The main advantage of arithmetic coding in this algorithm is that it contains a maximum of four symbols at any time. For instance, the encoder contains two symbols for subordinate passes, three symbols for dominant passes with no zerotree symbol and four symbols for dominant passes with zerotree symbol (the terms dominant pass and subordinate pass will be explained later). Because the maximum number of symbols is set to four, the occurrence of the possible symbols can be measured with less effort. This advantage lets the algorithm use a short memory to learn quickly and constantly changing symbol probabilities. Zerotree coding has a self-similarity property, which helps cost reduction for encoding significant coefficients. There is still a chance of dependency among the significant coefficients, though the coefficients are decorrelated before using the DWT decomposition.

3.2 EZW – The Algorithm

The output of the EZW encoder starts with the header, which contains information needed for the decoder to reconstruct the image. The basic information required by an EZW decoder is the size of the image, the number of levels used for the wavelet decomposition and the initial threshold value. The header can be avoided if we provide the correct information for the decoder. However, any incorrect information may result in a bad reconstruction of the image, as well as a higher PSNR value. For

the encoder to begin the encoding of the wavelets (which are already decomposed using the DWT), the threshold value is evaluated. The threshold value (T_0) must obey the rule $Xi < 2T_0$, where X_i represents all the transform coefficients. Thus, the maximum valued coefficient, X_{i_max} among all the transform coefficients is calculated.

$$X_{i_max} = \max(\max(N_N_image))$$

As stated previously, the EZW follows Successive Approximation Quantization, which uses a sequence of thresholds in the process of quantization, such as $T_0, T_1, \ldots T_N$, where $T_i = T_{i-1} / 2$. To reduce the complexity of the implementation, care should be taken that the threshold values are always a power of 2. This can be qualitatively explained as

$$T_0 = 2^{abs(\log_2(X_{i_max}))}$$

With this available information, the encoder can proceed to the main loop. The flowchart for the EZW encoder is shown in Figure 13. The EZW algorithm maintains two separate lists for the encoding, known as the Dominant List and the Subordinate List. For each threshold it passes through, the Subordinate Pass follows the Dominant Pass. The pseudo-code of the EZW encoder is given below. Initialization of the all variables is done as follows. The main loop calls both the Dominant Pass and the Subordinate Pass.

Initialize:

$k=0; T_0=2^{|\log 2(max_coeff)|}$

Dominant List = same as the image

Subordinate List = Null

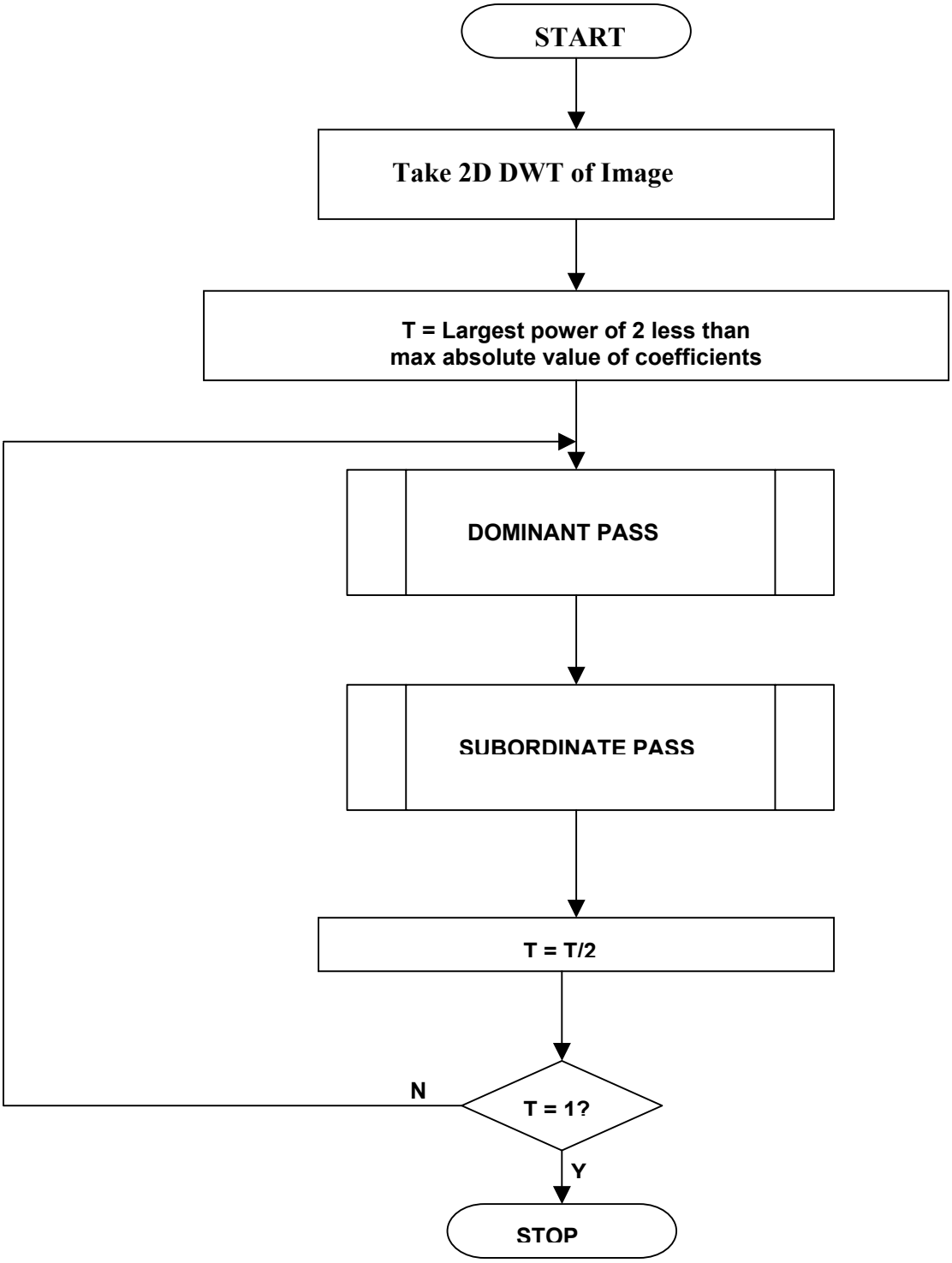


Figure 13 Flow-chart of EZW Encoder

Main loop

Do {

DominantPass (Image)

SubordinatePass (Image)

$T_{k+1} = T_k/2$

k=k+1

} while (Threshold = target lower value)

3.2.1 Dominant Pass

In the dominant pass, each wavelet-decomposed pixel value is compared with a threshold, and the significance of the pixel value is determined. As said earlier, the initial threshold value is chosen to be an integer power of two and less than the maximum valued pixel, i.e., $X_{i_max}/2 < T_0 (=2^n) < X_{i_max}$, where n is an integer. If the coefficient is larger than the threshold, a **P** (positive) is coded, whereas, if the coefficient is a negative number and the absolute value of the coefficient is larger than the threshold, an **N** (negative) is coded. When an insignificant value is found, it means the coefficient is smaller than the threshold. If the comparison of all the coefficient's descendants (or children) in the subsequent bands with the same threshold also insignificant, then it is feasible the parent pixel and all of its children can be encoded with only one symbol, **ZT** (zero-tree), thus achieving compression. If a coefficient is smaller than the threshold and it is not the root of a tree, then an **IZ** (isolated-zero) is coded. This happens when significant children exist for an insignificant parent. Figure 14 shows the flow chart for the encoding procedure of the dominant pass.

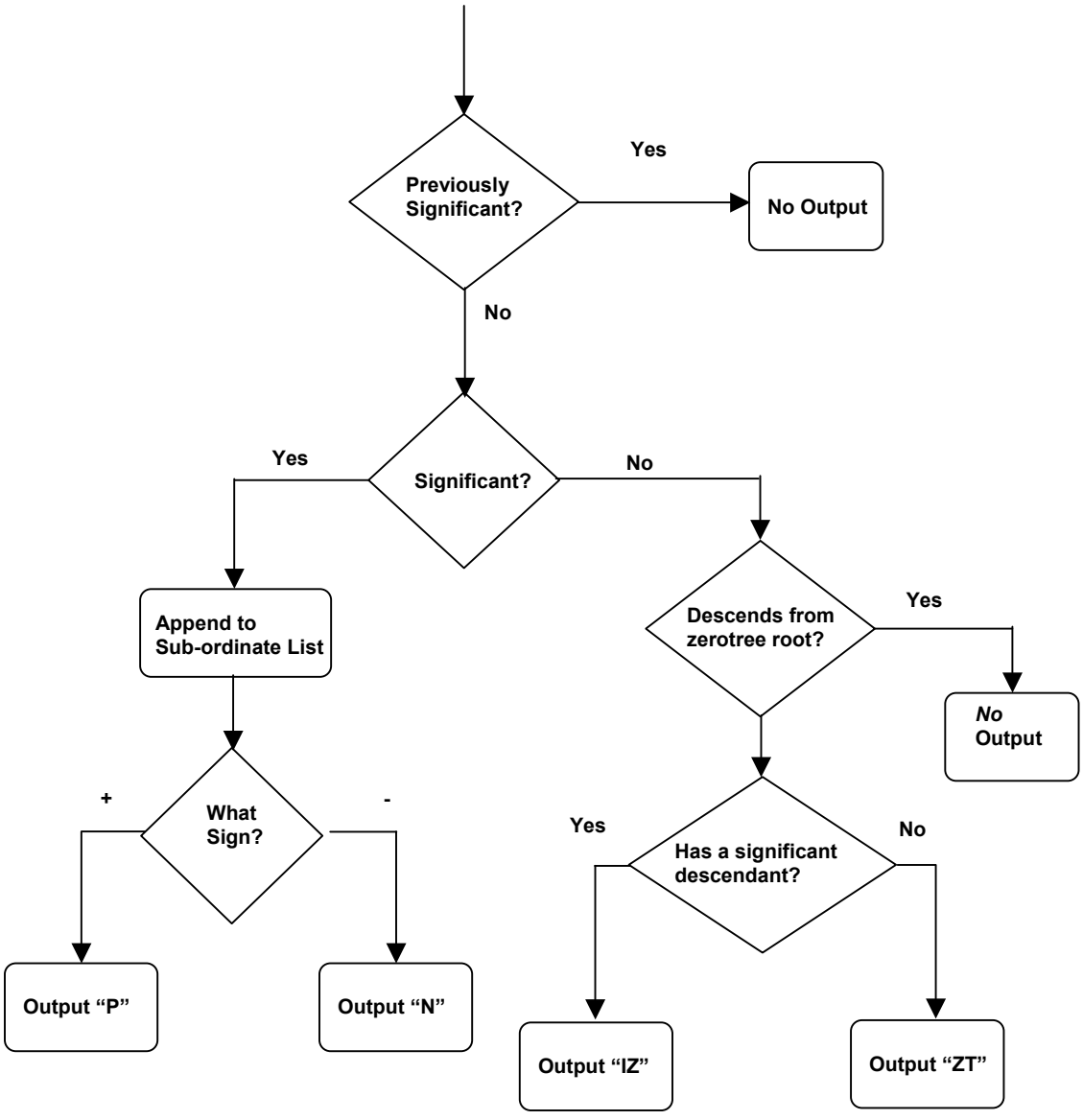


Figure 14 Flow-chart of Dominant Pass

According to the above explanation, in order to conclude that a coefficient is the root of a zerotree or an isolated zero, the encoder has to scan the whole tree. *This is the time-consuming operation in the encoding.* The encoder keeps track of the encoded symbols for coefficients to prevent re-coding of the coefficients that are already identified and encoded as zerotrees. If coefficients are found to be significant with a threshold, i.e., if they are coded as positive (P) or negative (N), then they are removed from the image and their positions are replaced by zeros. This will prevent them from being coded again with lesser threshold values in the next iterations. The absolute values of the removed significant coefficients are placed in the subordinate list. Dominant pass is also called significance pass, as the significance of the coefficients are determined in this pass. Any suitable scanning order can be used which will ensure that no child element is scanned before its parent element. Dominant pass can be represented in the form of pseudo code as shown below [3].

Dominant Pass:

For each entry X_i in the dominant list

If $|X_i| \geq T_k$ [i.e. X_i is significant]

If X_i is positive

Encode symbols ***P***

Else [i.e. X_i is negative]

Encode symbols ***N***

End if [for positive/negative confirmation]

Add $|X_i|$ to the subordinate list

Remove X_i from the Dominant list

Else [i.e. $|X_i| < T_k$, that means X_i is insignificant]

Case # 1: X_i is non-root part of a zerotree

Don't code – it is predictably insignificant

Case # 2: X_i is a root of the zerotree with all insignificant quad-tee elements – Encode symbol ***ZT***

Case # 3: X_i is a root of the zerotree with one or more significant quad-tee elements – Encode symbol ***IZ***

End if [for the significance]

Entropy coding of the symbols using Adaptive Arithmetic Coding [Optional]

Save the encoded bits.

End loop through the Dominant Pass

3.2.1 Subordinate Pass

The Subordinate Pass, also known as the Refinement Pass, is conducted on the subordinate list (containing the previously found significant coefficients) immediately after the Dominant Pass. The Subordinate Pass performs the pixel value quantization, i.e. assigns the pixel value a symbol, by which the decoder can roughly estimate the pixel value while reconstructing the image. Since the initial threshold is one-half the maximum pixel value of the image during the first Dominant Pass, the uncertainty of the significant value lies in the interval $[T_0, 2T_0]$. Thus, the first Subordinate Pass specifies only two ranges in which the significant value could lie: upper range, which is between $[3T_0/2, 2T_0]$ or lower range, which is between $[T_0, 3T_0/2]$. If the significant coefficient falls in the upper range, it is encoded as **H** or it is

encoded as **L**.

The flow chart of the Subordinate Pass is shown in Figure 15. The Subordinate Pass is not really an essential operation in terms of the reconstruction, as the decoder can reconstruct the image with the bits generated by the Dominant Pass. However, the Subordinate Pass will help to increase the quality of the reconstructed image by supplying adequate image data. The Subordinate Pass can be represented in the form of pseudo code, as seen below [3].

Subordinate Pass

For each entry X_i in the Subordinate List

If $X_i \in [3T_0/2, 2T_0]$

Encode H ("H" for "high")

Else [i.e., $X_i \in [T_0, 3T_0/2]$]

Encode L ("L" for "low")

End if

Entropy code H's and L's using Adaptive Arithmetic coding

Save the encoded bits.

End loop through the Subordinate List

3.3 EZW – an example

The example of a 2-scale wavelet transformation of an 8x8 image is used to explain the algorithm. The image values are shown in Figure 16. The initial threshold (T_0) is determined according to the equation, $T_0=2^{\lfloor \log 2\ (max_coeff) \rfloor}$, and so the first step is to find the maximum image value, seen to be 125 in Figure 3.5. Then the initial threshold can be set to 64. The Dominant List is actually the same as the image, which is an 8x8 array of pixel values. The Subordinate List is a one-dimensional

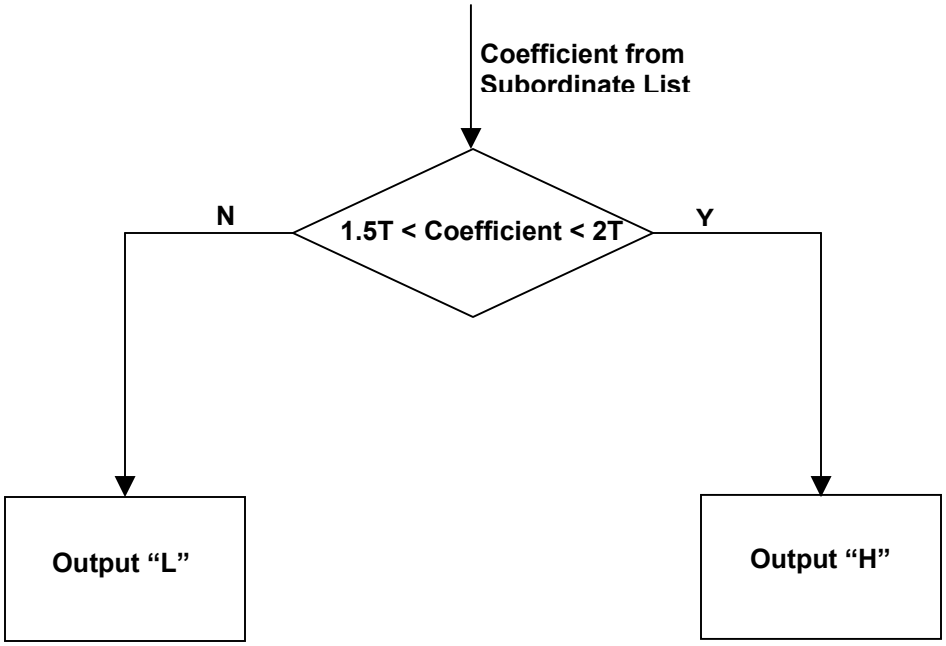


Figure 15 Flow-chart of Subordinate Pass

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 16 An 8x8 Sample Image

row matrix, initially a null matrix. The first Dominant Pass is then conducted using the initial threshold 64 and is explained as follows.

1) The first coefficient, 125, which is in level 3, subband LL_3, is greater than the threshold 64 and is positive. Therefore, a positive symbol **P** is coded. [See Figure 17]
2) The scanning order of the coefficients is –78, 15 and 9, which belong to the 3rd level subbands HL_3, LH_3 and HH_3, respectively. Compared to the threshold 64, -78 is greater but negative, and so **N** is coded. [See Figure 18]
3) The coefficient 15 is insignificant compared with coefficient 64. The 2nd level subband LH_2 coefficients {11, 10, 8, -6) are also insignificant compared to coefficient 64. However, the 1st level subband LH_1 has a significant coefficient 95, and so the root of the zerotree 15 is coded as insignificant zero, **IZ**. [See Figure 19]
4) The coefficient 9 is less than 64, and the next finer subband coefficients {4, 8, -5, 14} and {4, 6, -2, ... , 3, -4, 4} are also insignificant. Therefore, 9, the root of the zerotree is coded as **ZT**. [See Figure 20]
5) The scanning of the coefficients follows the order 99, -5, 1, -3 then 11, 10, 8, -6 and 4, 8, 5, -14. These coefficients's location can be represented as the second level subbands, which are HL_2, LH_2 and HH_2 [See Figure 21]

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 17 Step#1 of Dominant Pass

125	**-78**	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 18 Step#2 of Dominant Pass

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	**95**	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 19 Step#3 of Dominant Pass

125	-78	99	-5	7	13	-12	7
15	**9**	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 20 Step#4 of Dominant Pass

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 21 Step#5 of Dominant Pass

6) Since coefficient 99 is greater than 64 and positive, it is coded as **P**. The next four coefficients, -5, 1, –3 and 11, are coded as **ZT,** because they are not descendants of any other zerotree root and their own descendants are insignificant compared to coefficient 64. The preceding statement can also be illustrated as follows: because -78, the parent coefficient of –5, 1 and –3, was significant and 15, the parent coefficient of 11, was coded as isolated zero. [See Figure 22, also see Figure 18 and 19]

7) The coefficient 10 is less than 64, but it has a significant descendant 95 in the next generation. So, it is coded as isolated zero **IZ**. 8 and –6 are coded as **ZT**s (zerotree) as explained in the previous step. [See Figure 23]

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	10	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 22 Step#6 of Dominant Pass

125	-78	99	-5	7	13	-12	7
15	9	1	-3	-3	4	6	-1
11	**10**	4	8	5	-7	3	9
8	-6	5	-14	4	-2	3	-2
-5	9	-1	**95**	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 23 Step#7 of Dominant Pass

8) The next coefficients to be coded are 4, 8, -5 and –14. All four are insignificant compared to 64 and they are in the non-root part of the zerotree, as their ancestor 9 is the root of the zerotree. Therefore, all four coefficients are left un-encoded. [See Figure 24]

9) Until now, all the coefficients, except the coefficients in the subbands LH_1, HL_1, HH_1, are encoded. Usually, most of the higher frequency subband coefficients are not encoded at higher threshold values, as they are often descendants of roots of zerotree or usually insignificant. For the level-1 subbands (HL_1, LH_1 and HH_1), the encoder uses only 3 symbols (P, N, Z), because these subband coefficients do not have any descendents and cannot be the roots of a zerotree. The final significance map is shown in Figure 25.

During the first Dominant Pass of reconstruction, if the decoder sees a symbol **P** and already knows the initial threshold value to be 64, the decoder outputs 96, the midpoint of the range [64, 128], as the reconstructed value [see Table 3]. However, the actual value of the coefficient in the original image is 125. The difference between the original and the reconstructed coefficient is higher, and so EZW uses another pass i.e. Subordinate Pass to refine the encoding information of the already found significant coefficients. Subordinate Pass is performed immediately after each Dominant Pass. The following comments explain the first subordinate pass [see Table 4].

125	-78	99	-5	7	13	-12	7
15	**9**	1	-3	-3	4	6	-1
11	10	**4**	**8**	5	-7	3	9
8	-6	**5**	**-14**	4	-2	3	-2
-5	9	-1	95	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 24 Step#8 of Dominant Pass

P	N	P	ZT	Z	Z	*	*
IZ	ZT	ZT	ZT	Z	Z	*	*
ZT	IZ	*	*	*	*	*	*
ZT	ZT	*	*	*	*	*	*
*	*	Z	P	*	*	*	*
*	*	Z	Z	*	*	*	*
*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*

Figure 25 Step#9 of Dominant Pass

Table 3 First Dominant Pass

Scanning Order	Subband	Coefficient Value	Symbol	Reconstruction Value
1	LL_3	125	P	96
2	HL_3	-78	N	-96
3	LH_3	15	IZ	0
4	HH_3	9	ZT	0
5	HL_2	99	P	96
6	HL_2	1	ZT	0
7	HL_2	-5	ZT	0
8	HL_2	-3	ZT	0
9	LH_2	11	ZT	0
10	LH_2	10	IZ	0
11	LH_2	-8	ZT	0
12	LH_2	6	ZT	0
13	HL_1	7	Z	0
14	HL_1	13	Z	0
15	HL_1	-3	Z	0
16	HL_1	4	Z	0
17	LH_1	-9	Z	0
18	LH_1	95	P	96
19	LH_1	-3	Z	0
20	LH_1	2	Z	0

Table 4 First Subordinate Pass

Coefficient Absolute Value	Arithmetic Coding of the Symbol	Reconstruction value
125	1	112
-78	0	80
99	1	112
95	0	80

1) During the first dominant pass, a subordinate list is created containing only significant coefficients, which are encoded either as P or as N. Thus, for the above example, the subordinate list is {125, -78, 99, 95}.

2) Two intervals, upper and lower, exist for each subordinate pass, depending on the threshold value. For threshold 64, the upper interval is defined between [96, 128], and the lower interval between [64, 96].

3) The first coefficient of the subordinate list, 125, belongs to the upper level, and so it is encoded as **H**. The reconstruction value is the center of the upper interval, or 112.

4) The next coefficient is 78, which is placed in the lower interval and encoded as **L**. The reconstruction value is the center of the lower interval, or 80.

5) The third entry, 99, is encoded as **H** and has a reconstruction value of 112. Finally, the last entry, 95, is encoded as **L**, and its reconstruction value is set to 80.

Notice that the reconstruction value after the subordinate pass of the coefficient 95 is changed from 96 to 80, which results in an increase of reconstruction error from 1 to 15. However, the uncertainty interval is decreased from 32 (64<96<128) to 16 (64<80<96), which will ensure overall improvement of reconstruction error. The subordinate list from the first subordinate pass is carried over to the second subordinate pass, and the significant values generated are placed next to the previously found significant values. In addition, the subordinate list's coefficients

are reordered based on decreasing order of the reconstruction values. Initially, the subordinate list follows the same order as the scanning order, which is {125, 78, 99, 95}, and the reconstruction values of the respective coefficients are given as {112, 80, 112, 80}. After the first subordinate pass, the reconstructed values, in descending order, are {112, 112, 80, 80}. They make coefficient 99 precede coefficient 78, and so, the new order for future subordinate passes is {125, 99, 78, 95}. Notice that coefficient 99 still precedes coefficient 78, which is smaller, because the decoder considers both the coefficients alike since their reconstructed values are same.

3.4 Software Implementation

It was discovered during an Internet search that the EZW algorithm was developed as a course project by a group of students at Rice University, TX, in 1999 [4]. The documentation of their work was very limited. Even though many enthusiastic researchers in the field of image compression using wavelets have already developed the software implementation of this algorithm using MATLAB, C, and C++, the present research work takes its inspiration principally from the aforementioned course project. The course project source code was written in MATLAB and was not bug-free, as it was generated using older versions of the software package available half a decade ago. However, most of the MATLAB functions were corrected and updated to comply with the latest version of the software package used. Quite a few functions were added to achieve some specific functionality, and few of the existing functions were removed. Compilation and validation were done using the MATLAB 6.1 package. The following subsection will give an overview of how the implementation was done.

3.4.1 EZW specifications

According to the EZW algorithm, the image has to be decomposed using the Discrete Wavelet Transform. A system-defined, built-in DWT function was used here, as the implementation of the DWT is itself very complicated. The quantization and the compression blocks were implemented manually. No particular order for scanning of the wavelet coefficients was stated in the original EZW specifications, and so the Morton-Scan order shown in Figure 26 was used here. For the compression, the Adaptive Arithmetic Coder was used. The basic idea behind the arithmetic coder is to represent the symbols in binary numbers. Arithmetic coding achieves the compression by the probabilities of the occurrence of the symbol, but the simple Arithmetic Coding followed here was not really concerned about the probabilities of the symbols. The arithmetic coding procedure is briefly explained in Table 5 and Table 6, respectively, for the dominant pass and subordinate pass.

3.4.2 Deviations from EZW specifications

In the original EZW specifications proposed by Shapiro [1], the subordinate list should be rearranged depending on the tentative reconstructed values after each subordinate pass, so that the future subordinate pass uses the renewed subordinate list. The sorting of the subordinate list is found to have little influence on the quality of the reconstructed image. When any efficient sorting algorithm is used, the execution time of the algorithm depends on the number of elements, N, to be sorted. If the number N is large, the execution time of the sorting algorithm will show an effect

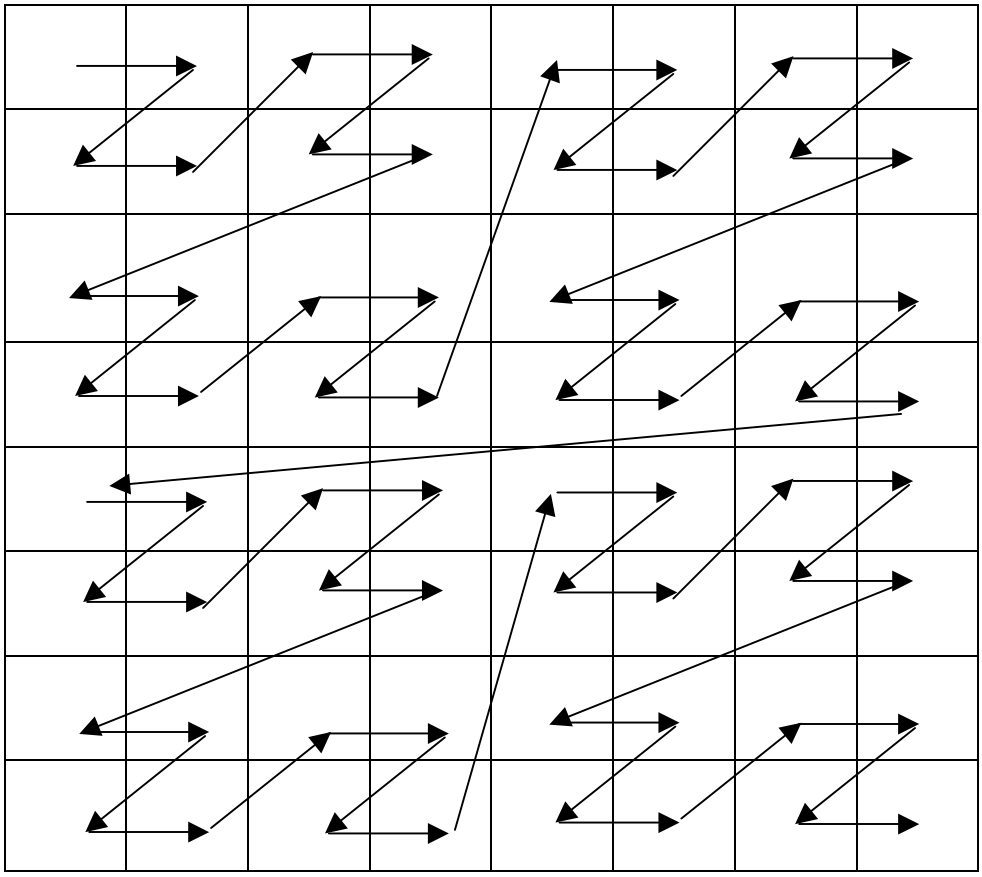


Figure 26 Morton-Scan Order

Table 5 Arithmetic Coding of Symbols for Dominant Pass

Dominant List Symbols	Binary Representation
P	“11”
N	“10”
IZ	“01”
ZT	“00”

Table 6 Arithmetic Coding of Symbols for Subordinate Pass

Subordinate List Symbols	Binary Representation
H	“1”
L	“0”

on the overall execution time of the encoder. However, because of the meager improvement achieved through the sorting of the subordinate list in the EZW, it was not implemented in the present work [4].

3.4.3 Decoding the Bitstream generated by EZW

The decoding process follows the same steps as the encoder. The flowchart of the decoder is shown in Figure 27. The decoder also uses two passes, Dominant Pass II and Subordinate Pass II, during the reconstruction process, similar to the dominant pass and subordinate pass of the encoder. In the beginning, the image to be reconstructed is initialized to all zeros, and then the encoded bitstream is passed through the dominant pass II. The bitstream comes with a header that contains the essential information needed for the decoder, such as initial threshold value, image dimensions, and the number of levels used for the DWT decomposition. As the bitstream is in binary 1's and 0's, the decoder reads two bits from the bitstream and turns them into symbols. If the decoder finds a positive or negative symbol, it places the reconstruction value of that particular pass, which is 3/2 times the threshold value, at the corresponding location. If the symbol is zerotree root (ZT) or insignificant zero (IZ), the corresponding coefficient positions are filled with the suitable values. The Dominant Pass II ends after scanning all coefficients of the image. The scanning order must be same for both encoder and decoder. Subordinate Pass II reads one bit from the bitstream. As discussed in earlier sections, if the bit is "1", then the corresponding coefficient's reconstruction value is reorganized to a higher value; if "0", then to a lower value. The term "embedded" is justified in the decoding process, as the decoder

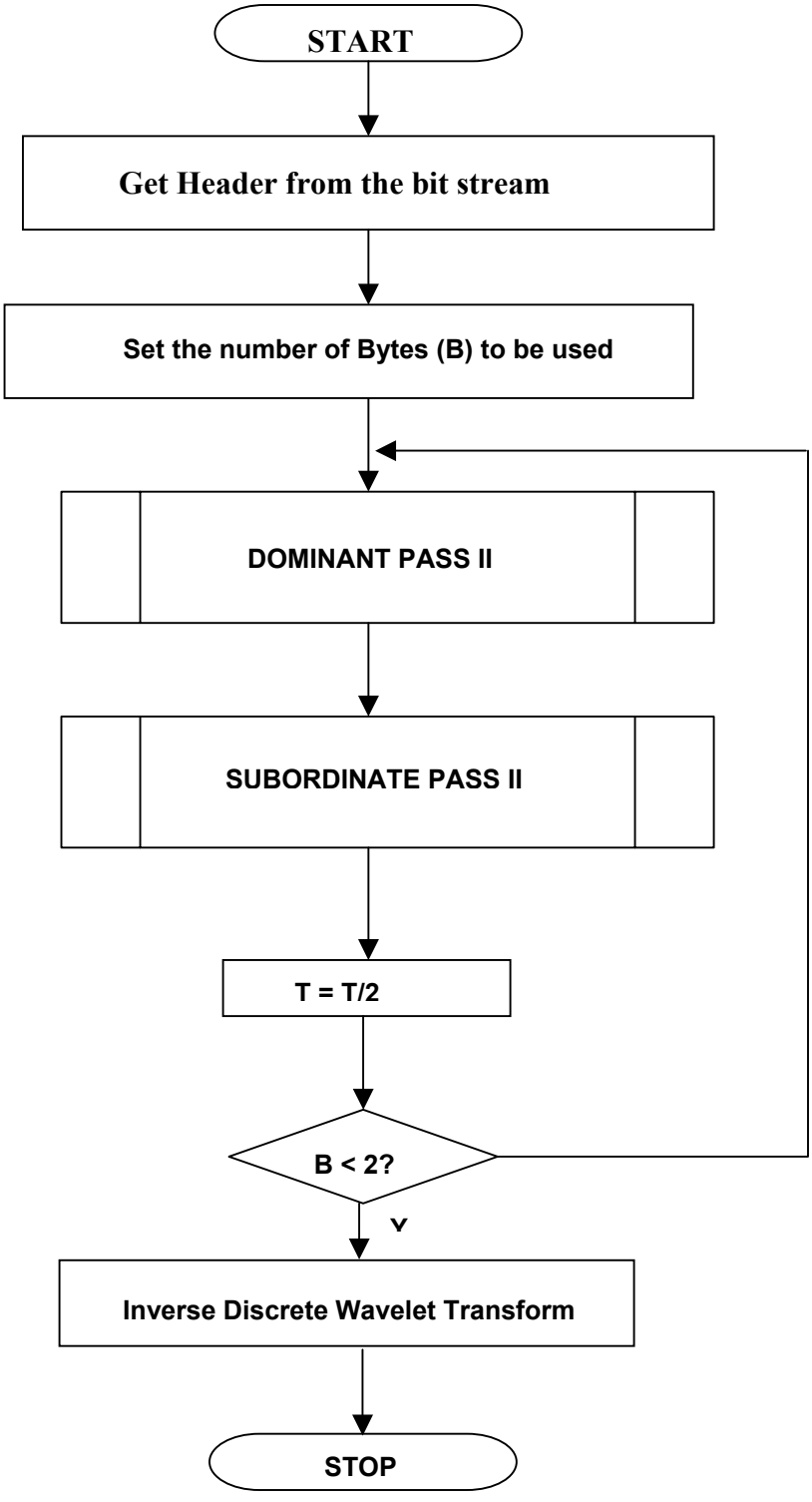


Figure 27 Flow-chart for the Decoder

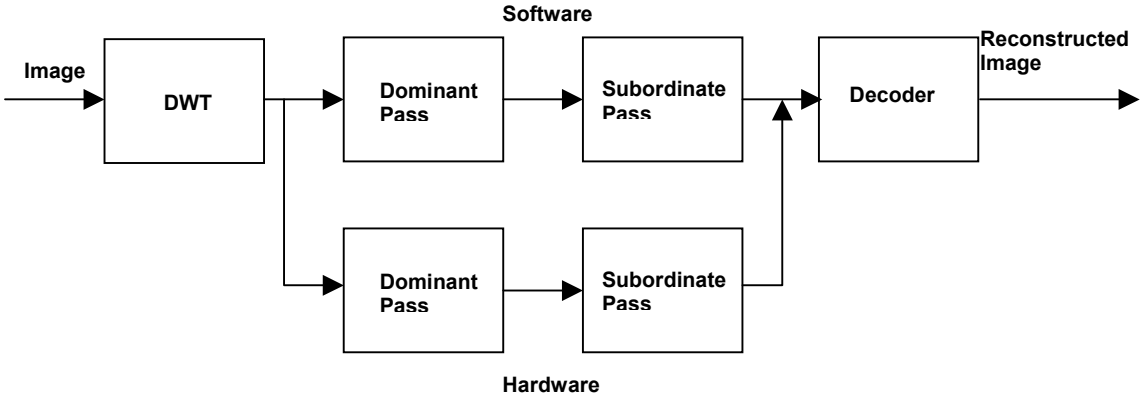


Figure 28 Block Diagram of the EZW Implementation

can stop at any time. The quality of the reconstructed image is directly proportional to the number of bits decoded. If the image is encoded until the least value is recognized, and if the bitstream is decoded until the last bit, then the resulting reconstructed image will be the same as the original image.

3.5 Hardware Implementation

The block diagram of the implementation (software and hardware) of the EZW algorithm is shown in Figure 28. The following subsections will give a brief explanation of how it is implemented in FPGA.

3.5.1 Explicit Design Flow of the Pilchard RC Platform

The Pilchard Reconfigurable Computing Platform, explained in the previous chapter, was used for hardware implementation. The design flow described in

this section is explicit for the above stated platform. Figure 29 depicts the design flow for implementation of any design on the Pilchard RC platform. The behavior of the encoder is coded using VHDL. The process of testing the module's functionality before translating it to the circuit level design through synthesis is technically known as pre-synthesis simulation. These simulations are performed with the aid of a "test-bench" written in VHDL. The VHDL simulator used here is Mentor Graphic's Modelsim® version 5.8d. Modelsim allows viewing all the signals and I/O ports with the help of a waveform viewer during the simulations. This helps debug the design. After successful pre-synthesis simulations, the design moves to the next step, synthesis. In this step, the design is synthesized using synthesis tools targeting to Xilinx® Virtex™ 1000E FPGA. Two different synthesis tools, Synopsis® FPGA Compiler II and Synplicity® Sinplify-Pro, were used. The inputs to these synthesis tools are VHDL modules, and the outcome is a structural netlist in EDIF format, which contains the gate level circuit descriptions. Some of the VHDL descriptions are not synthesizable. In that case, the synthesis tool immediately gives an error message. The synthesis tools partially report timing violations because the gates are not yet completely routed. All these problems have to be fixed before moving to the next step, i.e., Place and Route (PAR). Xilinx® ISE tools are used for PAR and Figure 30 shows the sequence of operations during PAR. An EDIF file generated in the synthesis and a pin constraint file (PCF) are the inputs of the PAR tools. First, the input information is passed through NGDBUILD, where the EDIF format is translated to Xilinx's Native Generic Database (NGD) format, then, the NGD file is mapped to primitives inside the particular target FPGA, which is Xilinx's Virtex™ 1000E. The output file format after the mapping

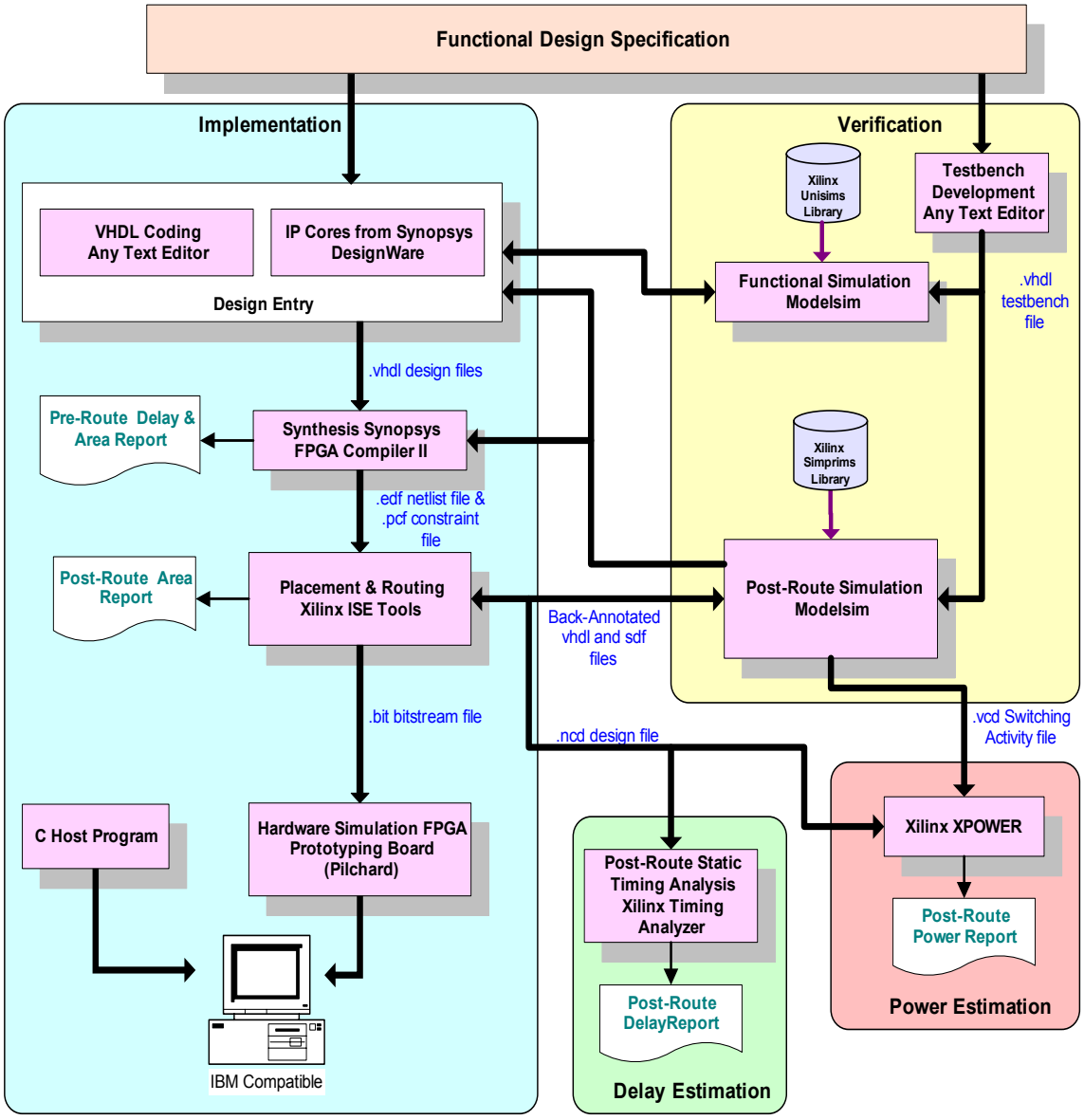


Figure 29 Explicit design Flow for Pilchard RC
(Courtesy: Dr. Chandra Tan)

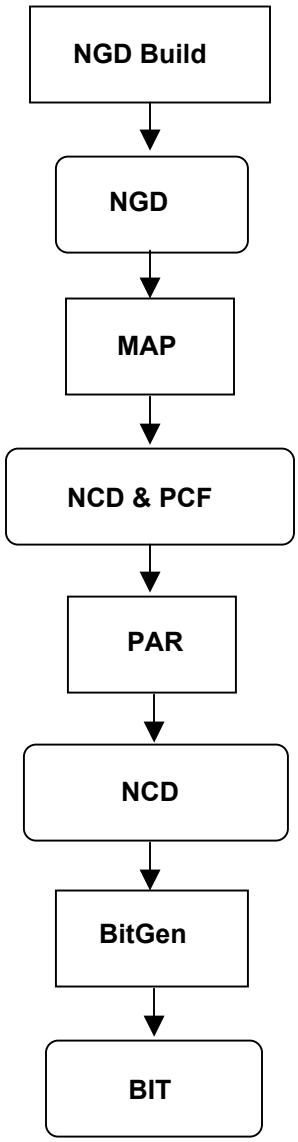


Figure 30 Flow of Design in the PAR tools

operations is Native Circuit Description (NCD) format, which is a physical representation of the design mapped to the components in the Xilinx FPGA. The PAR takes the NCD file as input, place and routes the design, and outputs another NCD file. After this step the timing constraints are revealed. If the design fails to meet the required timing, then the design has to be changed. After successful PAR, the design moves to the bitstream generator (BitGen). PAR tools can generate a post route simulation netlist in HDL and a timing file with .SDF (Standard Delay Format) extension if the user desires to perform post-layout simulations. BitGen takes a fully routed NCD file as its input and produces a configuration bitstream, a binary file with a .BIT extension. The BIT file contains both all of the configuration information from the NCD file which defines the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device [17].

3.5.2 Design Details

The EZW encoder consists of four main modules: encoder, parith, DP-RAM and pcore. The hierarchy of the modules is shown in the Figure 31. Each of the main components were implemented using VHDL and were simulated using Modelsim. Simulations were performed at each level of the hierarchy and the results were compared with the known values to confirm that the project is heading in the right direction. The RAM module was generated using Xilinx's Coregen. The Encoder uses a package that contains descriptions of all arithmetic operations done in the module. Parith, the next higher-level module, connects the controller and the encoder.

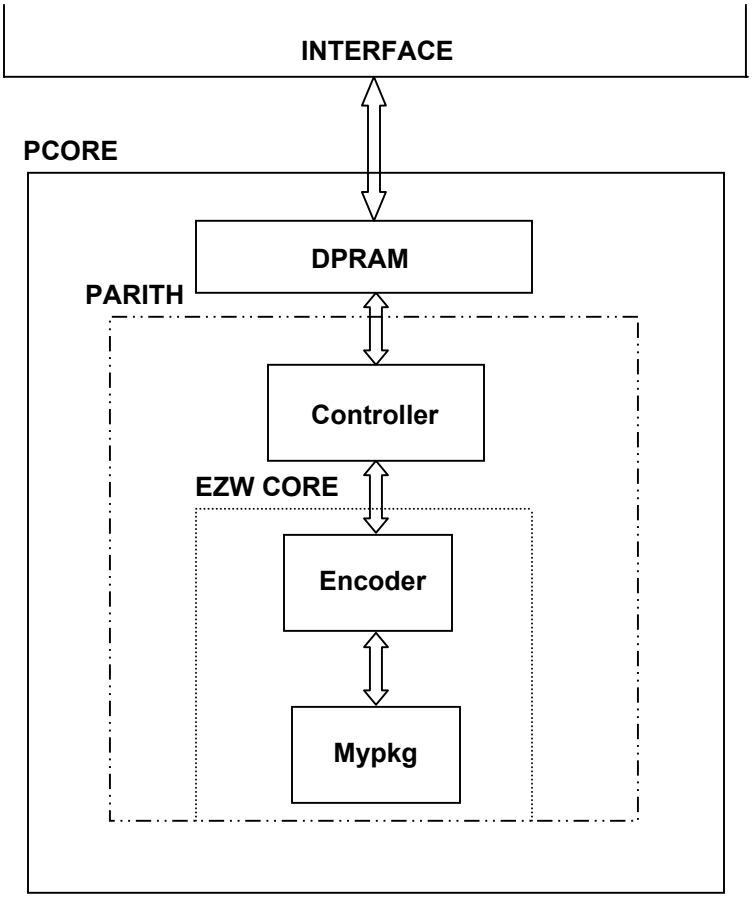


Figure 31 Hierarchy of the modules

3.5.2.1 Encoder

Genuine encoding happens in this module. The inputs are standard logic vectors, but the logic inside actually utilizes decimal numbers. Thus, several binary-to-decimal converters were written into the package part of the module, which can be called up by the entity part of the module. The outputs, too, are standard logic vectors, and so decimal-to-binary converters were also written into the package part of the module. VHDL needs synchronization of the process, unlike high-level software programming. That means the logic cannot be implemented sequentially, as the logic will be translated to physically realizable circuits later. This brings up the Finite State Machine (FSM) modeling. Finite State Machine (FSM) modeling is very common in RTL design and, therefore, deserves special attention. Figure 32 shows a sample state machine, and the VHDL representation of the FSM can be seen in Figure 33. The Finite State Machine of the ENCODER is shown in Figure 34 and the brief information about the operations of each state can be seen in Table 7.

3.5.2.2 Controller

The controller is the connecting unit of the DP_RAM and the encoder. It controls the action of the encoder, supplying the inputs and collecting results. The intermediate-level top module PARITH wraps up the CONTROLLER and the ENCODER. The Finite State Machine of the CONTROLLER is shown in Figure 35, and brief information about the operations of each state can be seen in Table 8. Pcore is top module of the design, which hooks up dual-port RAM and Parith (Controller + Encoder) modules. This concludes the explanation of the modules.

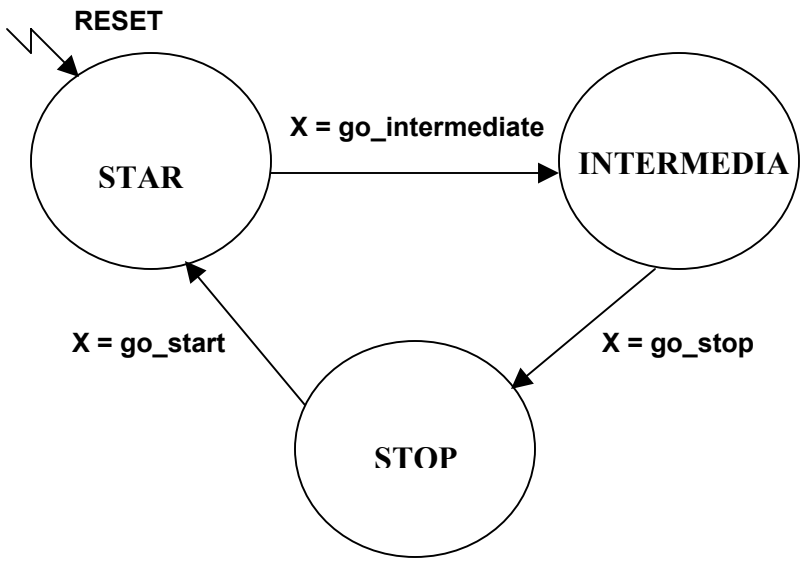


Figure 32 A sample FSM

```
FSM_SAMPLE: process (CLK, RESET)
begin
 if RESET='1' then
       STATE <= START  ;
   elsif CLK'event and CLK='1' then
      case  STATE  is
            when  START   => if  X=GO_INTERMEDIATE  then
                        STATE <= INTERMEDIATE  ;
                     end if ;
            when  INTERMEDIATE  => if  X=GO_STOP  then
                        STATE <= STOP  ;
                     end if ;
            when  STOP    => if  X=GO_START  then
                        STATE <= START  ;
                     end if ;
            when  others  =>  STATE <= START  ;
         end case ;
   end if ;
end process FSM_SAMPLE ;
```

Figure 33 VHDL Representation of a sample FSM

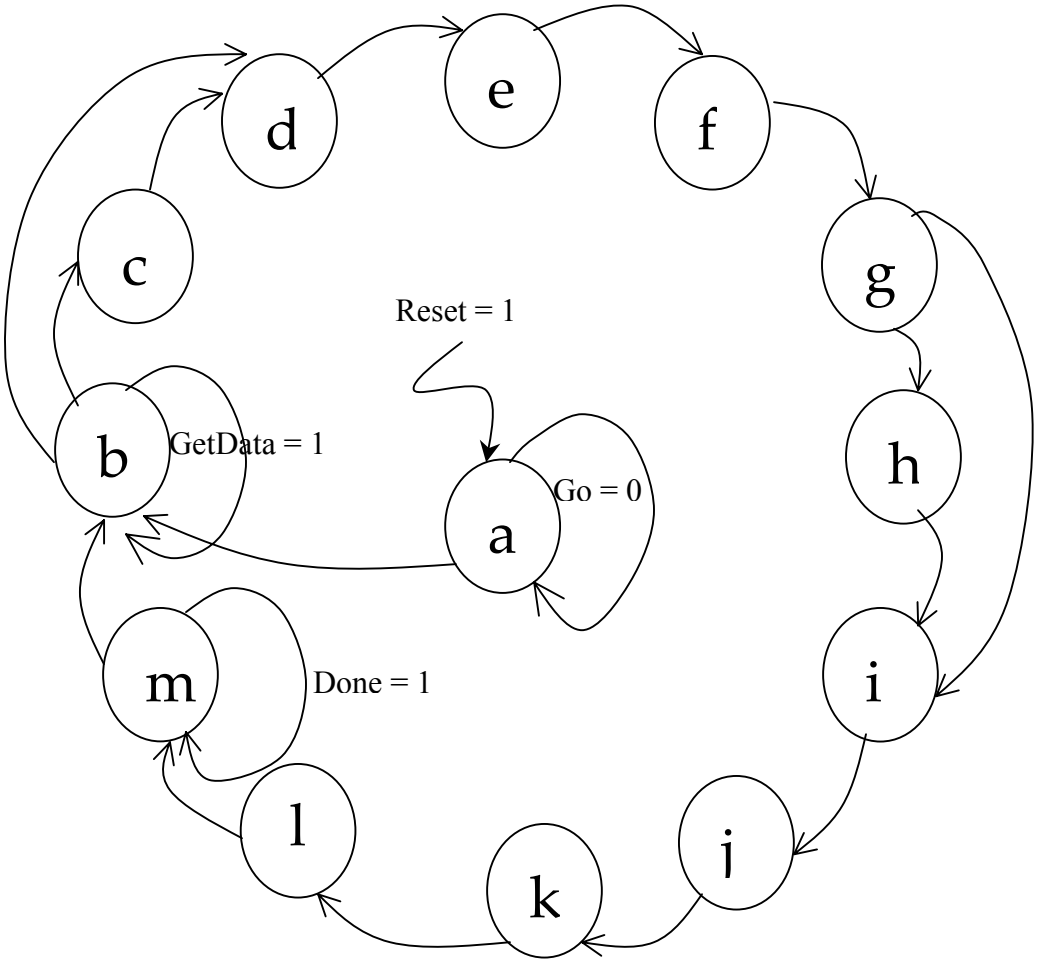


Figure 34 FSM of the Encoder

Table 7 Detailed operations of each State of the Encoder

State	Operation
a	If (go =1) then initialize all the required signals DominantList, SubordinateList, SignificanceMap State = a; Else, wait for go signal to become 1
b	If (GetData =0) then State = c or State=d (for first DominantPass); Else, compare the input data with initial threshold update the SignificanceMap and Subordinate List.
c	Match the SignificanceMap with DominantList; State = d;
d	Finding the Descendants (coarser subband) i.e., Low-Low State = e;
e	Finding the Descendants (finer subbands) i.e., High-Low; Low-High; High-High; State = f;
f	Finding the Ancestors (coarser subband) and finalizing Zerotrees State = g;
g	Finding the Ancestors (finer subband) and finalizing Zerotrees State = h or State =i;
h	Check if the coefficient is already coded or not. Previously coded coefficients will not be coded again. State = i;

i	If (|Coeff |> T)
	If positive output “11”;Else output “10”;
	End if;
	Else
	If root of zerotree Output “00”Else output “01”
	End if;
	State=j;

j	If (3T/2 < Coefficient < 2T)
	Output ‘1’; Else, output ‘0’
	End if;
	State=k;

k	Store the results in a buffer; the outputs are chunks of 16 bits. State=l;
l	Correct the Dominant List; Remove the coefficients that are found significant; State=m;
m	Update the information; T= T/2; If (T>1) State = a; Else State=State; End if;

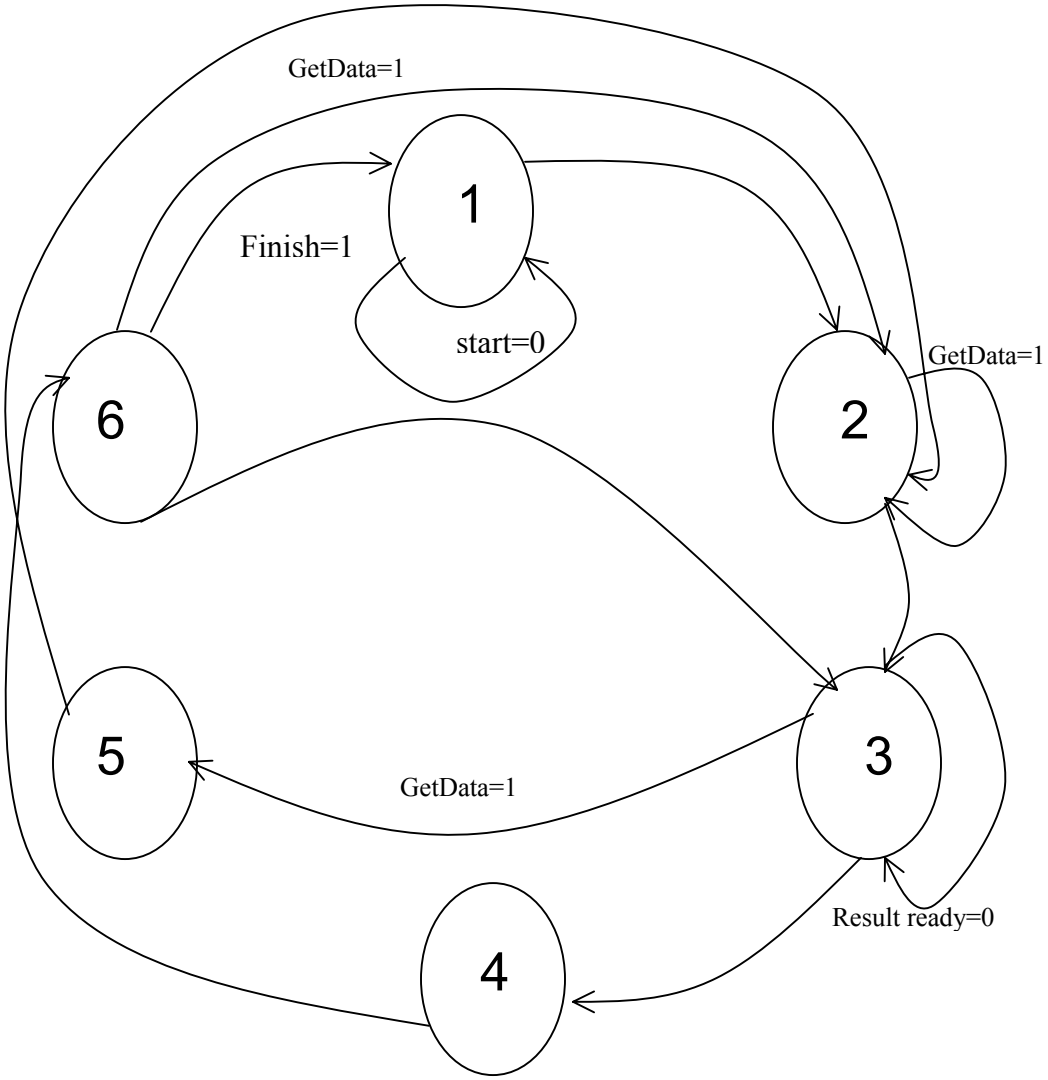


Figure 35 FSM of the Controller

Table 8 Detailed operations of each State of the Controller

State	Operation

1	If (start = 1)
	Initialize Read_Address
	Initialize Write_Address
	State=2;
	Else
	State=1;
	End if;

2	If (GetData=0) then
	State=3; reset the Read_Address to initial value;
	Go signal becomes 0;
	Else
	Go signal will be released for the ENCODER
	Read data from the RAM;
	update the Read_Address;
	State=2;
	End if;

3	If (result_ready=1) then,
	State=5;
	Elseif (GetData=1) then,
	State=4;
	Else
	State=3
	End if;

4	Disable the Write State=2;
5	Update the Write_Address; Write result to the RAM; State=5;

6	If (Done=1) then,
	Write the final result to the RAM;
	State=5;
	Elsif (GetData=1) then,
	State=2;
	Elsif (Finish=1) then,
	State=1;
	Else
	State=3;
	End if;

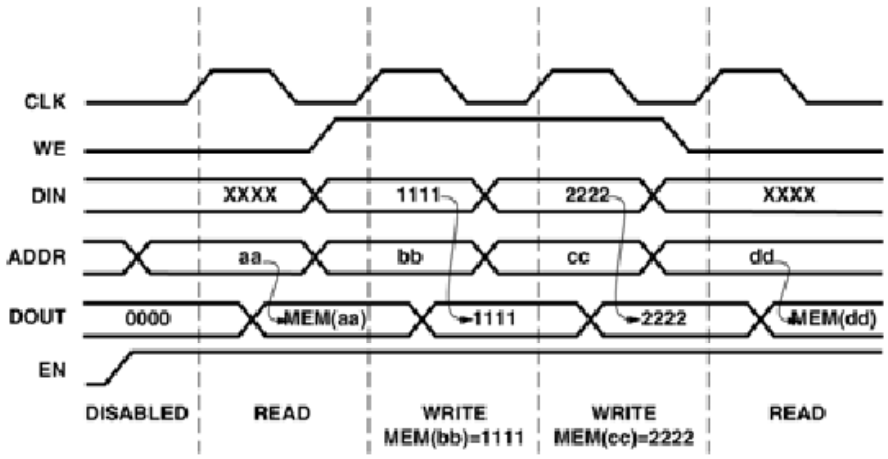


Figure 36 Read/Write Operations of Xilinx Dual-Port RAM [17]

3.5.2.3 Dual Port – RAM

The main advantage of using Dual-Port RAM is that read and write operations can be performed at the same port. The dual-port RAM used here is generated by the Xilinx® Core Generator™. The RAM module is implemented on-chip by mapping the RAM design to physically separate the RAM blocks, thus, the CLBs inside the FPGA are not consumed by it. A picture explaining read-write operations of the DPRAM is shown in the Figure 36. It takes two clock cycles to read from the RAM and one clock cycle to write to the RAM. Thus, to keep the synchronization intact, care should be taken while reading or writing from the RAM. When reading from the RAM, the controller issues the address of the data location first, and the relevant data is received after two clock cycles. When writing to the RAM, the controller issues the address and 'write-mode' is enabled. Data is updated in the next cycle. The 'write-mode' must be disabled in the subsequent cycle.

3.5.3 Pre-Synthesis Simulations

The pre-synthesis simulations are performed at all the levels of the design. Test-benches were generated for each level. Initially, the encoder was tested for the correctness, and then it was wrapped up with the controller and the DP-RAM. The top-level module of the design is the pcore module. The encoder's simulation waveforms are shown in Figure 37 and Figure 38. The pcore's simulations results are in Figure 39 shows I/O signals and Figure 40 showing done signal becoming 1 at 317400ns.

3.5.4 Synthesis

The synthesis of the VHDL descriptions was performed following successful simulations. Synthesis of the design was done using appropriate synthesis tools.

3.5.5 Place and Route (PAR)

Xilinx ISE tools were used for Place and Route. The layout of the design can be viewed at this step. Figure 41 show the layout generated Synplify Pro.

3.5.6 Bit file generation

If the design meets the timing constraints after PAR step, the design is ready to be downloaded on to the FPGA. A .bit file is generated using Xilinx ISE tools. This step also performs the Design Rule Check on the layout of the design (DRC).

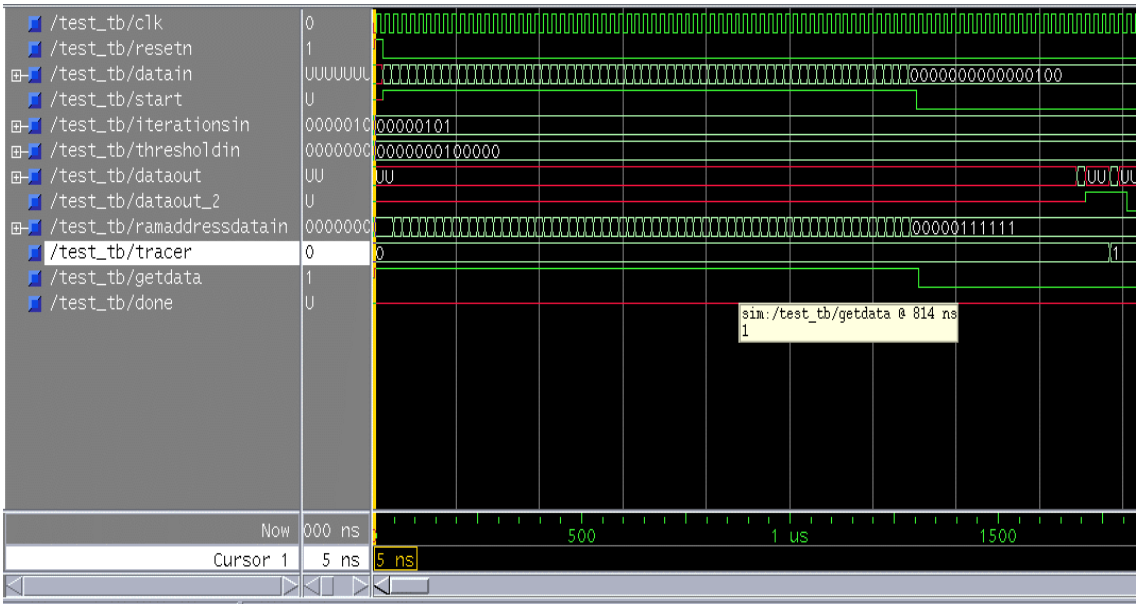


Figure 37 Waveform indicating inputs of encoder module

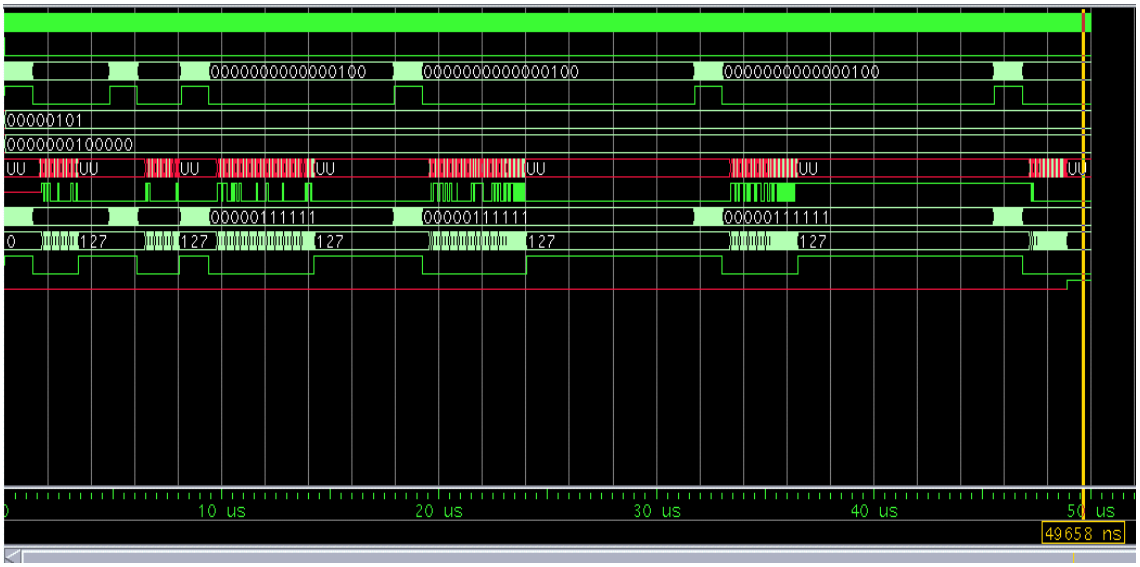


Figure 38 Waveform showing done signal becoming 1 at 49558 ns

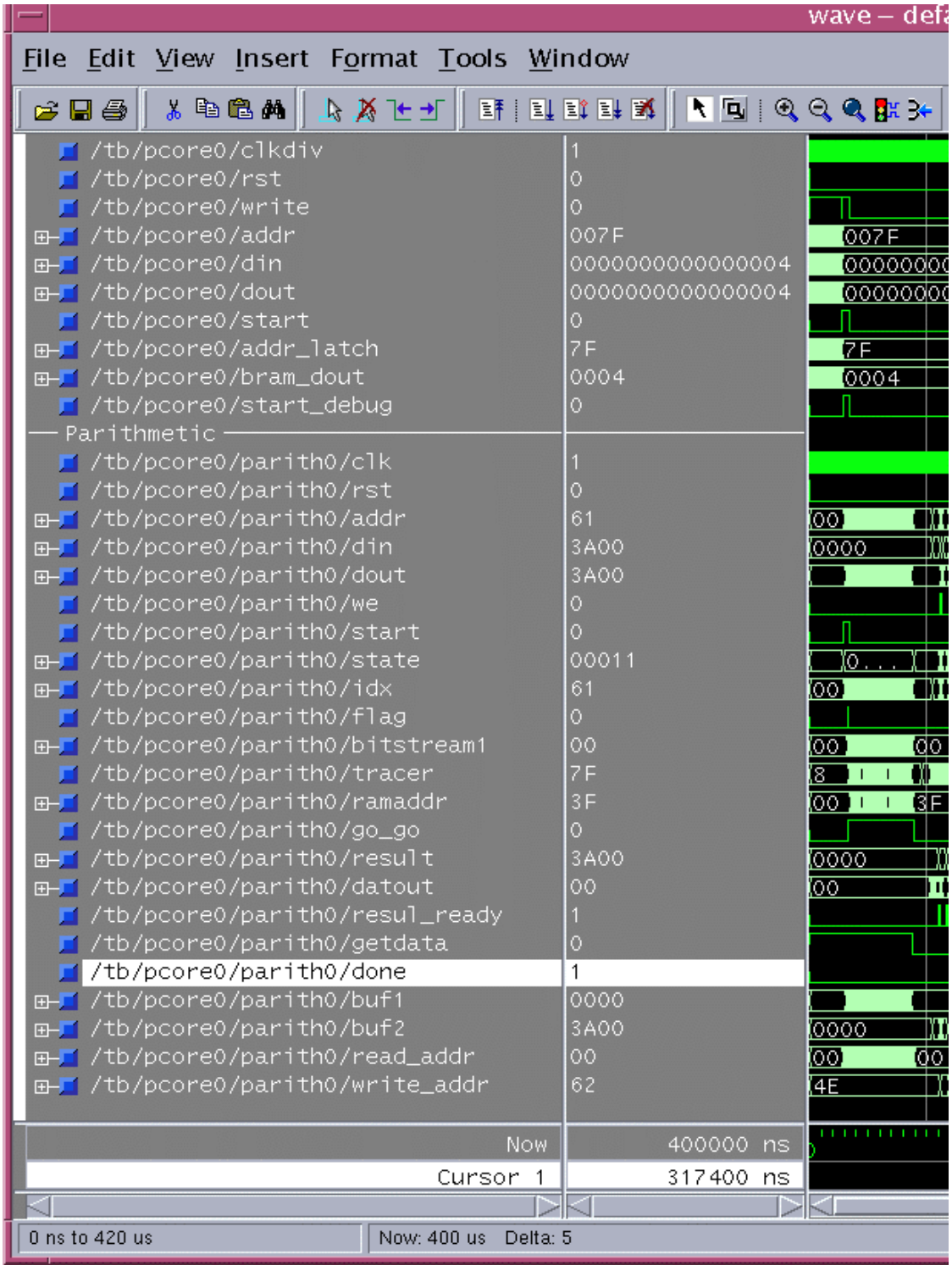


Figure 39 Waveform showing parith and pcore signals, and start becoming 1

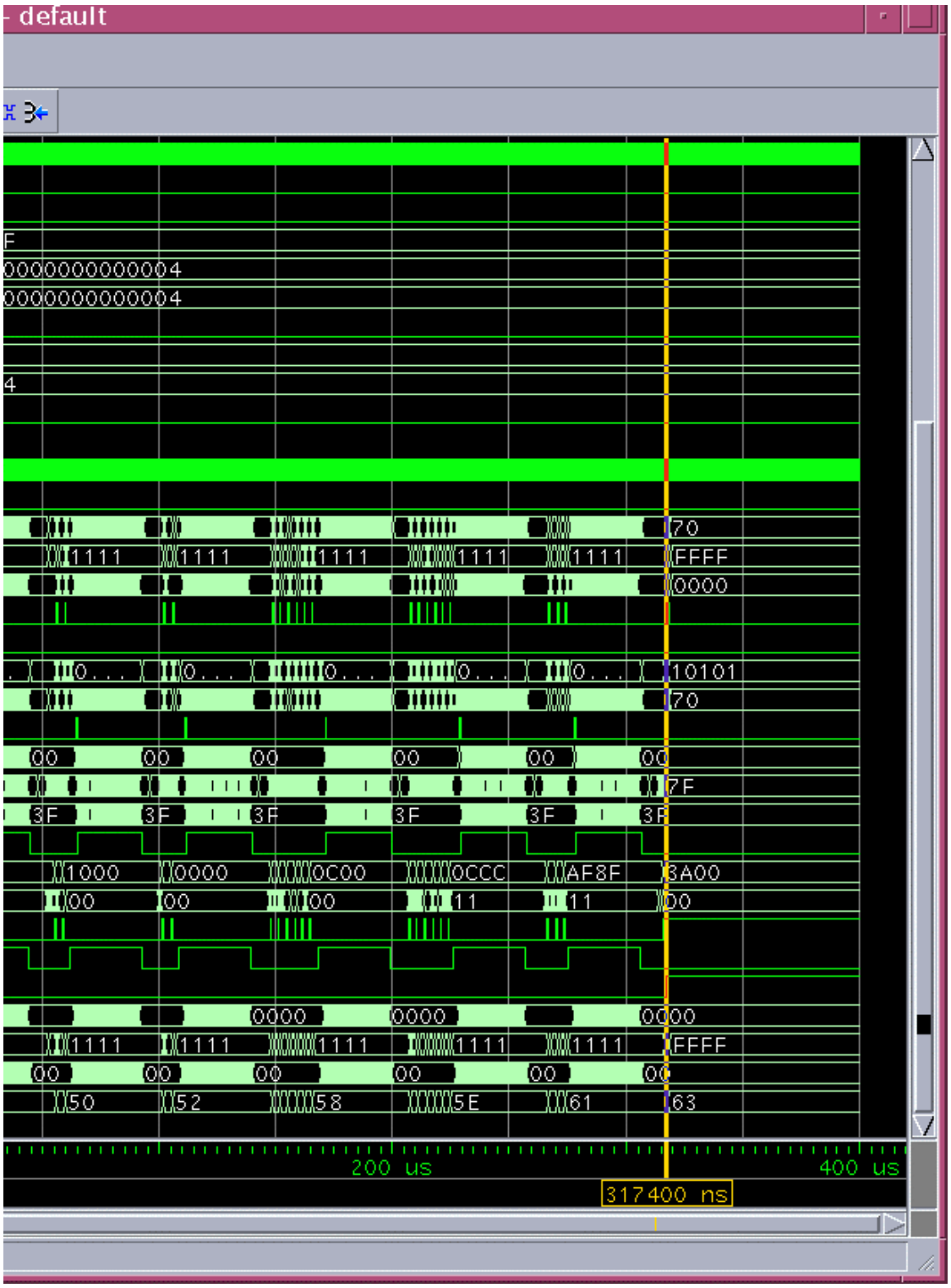


Figure 40 Waveform indicating done signal becoming 1 at 317400

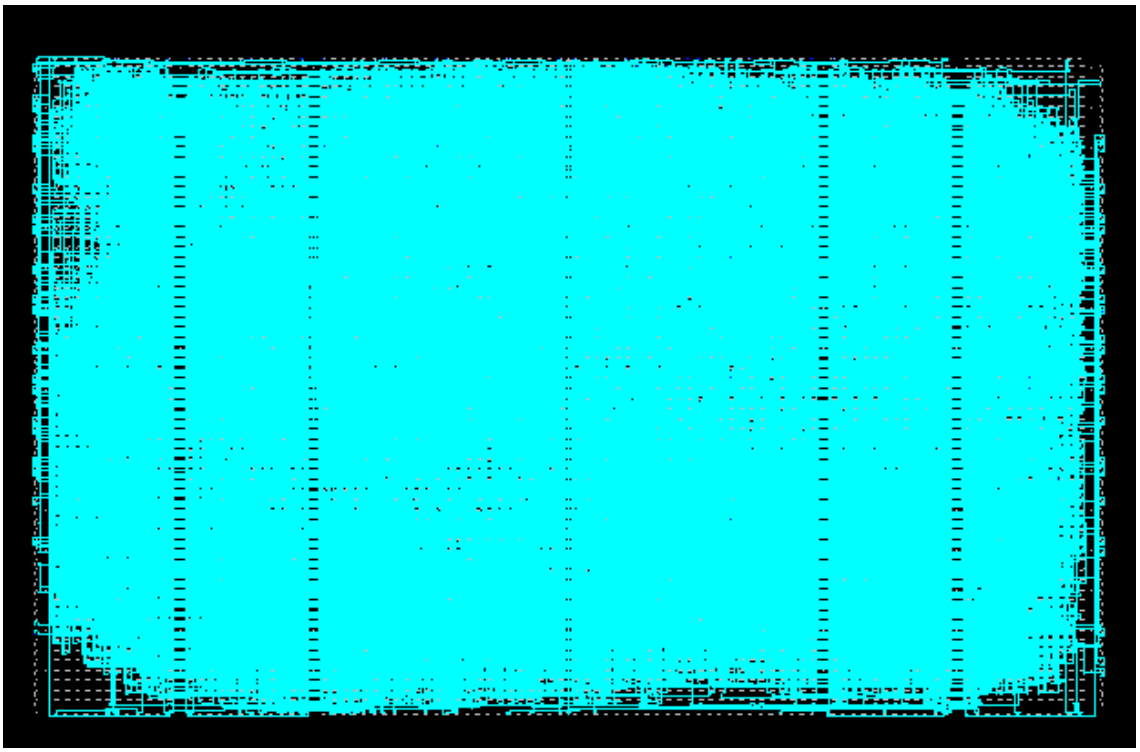


Figure 41 Layout of the Design using Synplify- Pro

CHAPTER 4
RESULTS

4.1 Software

The EZW algorithm was applied to three different test images, Lena, Barbara and Goldhill. The results obtained were tabulated. The entire input image files used here are in PGM format, which stands for Portable Gray Map. The PGM format is a lowest common denominator grayscale file format. The following subsections explain the procedure in detail.

4.1.1 Test Image – Lena

The original Lena image in PGM format is shown in Figure 42. First, with the help of Daubechies Wavelet Filters, the input image is transformed into wavelet coefficients using Discrete Wavelet Transform (DWT) at six levels. The decomposed image is shown in Figure 43. The Encoder has generated a bitstream file containing 42848 bytes, or at a bit rate of 0.1634 bpp (bits per pixel). The Decoder is able to recognize the bitstream file and can reconstruct the image using any target bit rate, i.e., using any number of bytes out of the total 42848 bytes. Various images were reconstructed using an assortment of number of bytes. At a compression ratio 1024 to 1, the image was not recognizable. However, at the compression ratio of 512 to 1 the image could be recognized even though the image quality was poor. Still, the compression ration of 512 to 1 is very high compared to the conventional block coding (DCT-based). At such high compression, any DCT-based compression techniques even fails to reconstruct the Image.

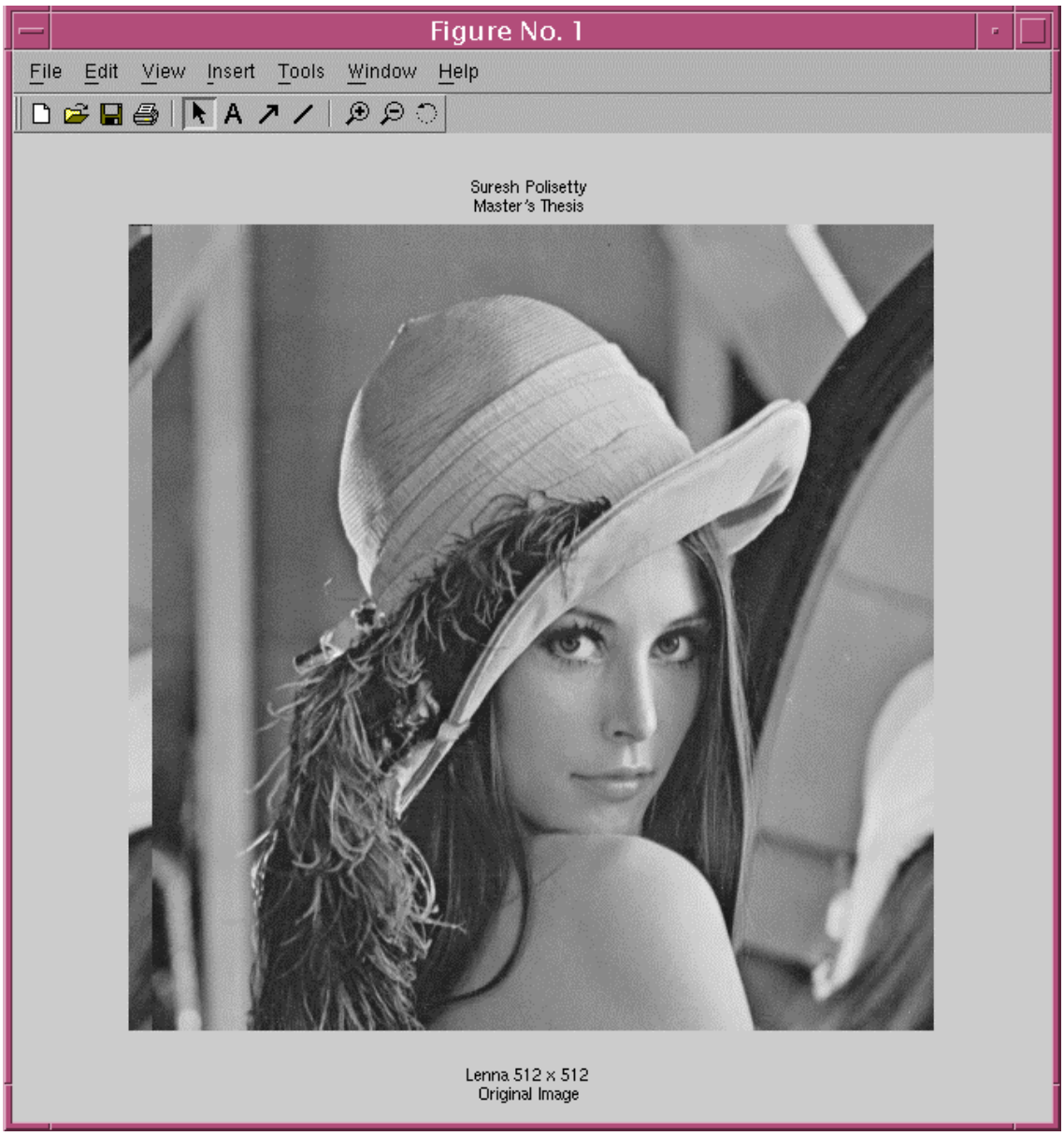


Figure 42 Test Image – Lena 512 x 512 Original

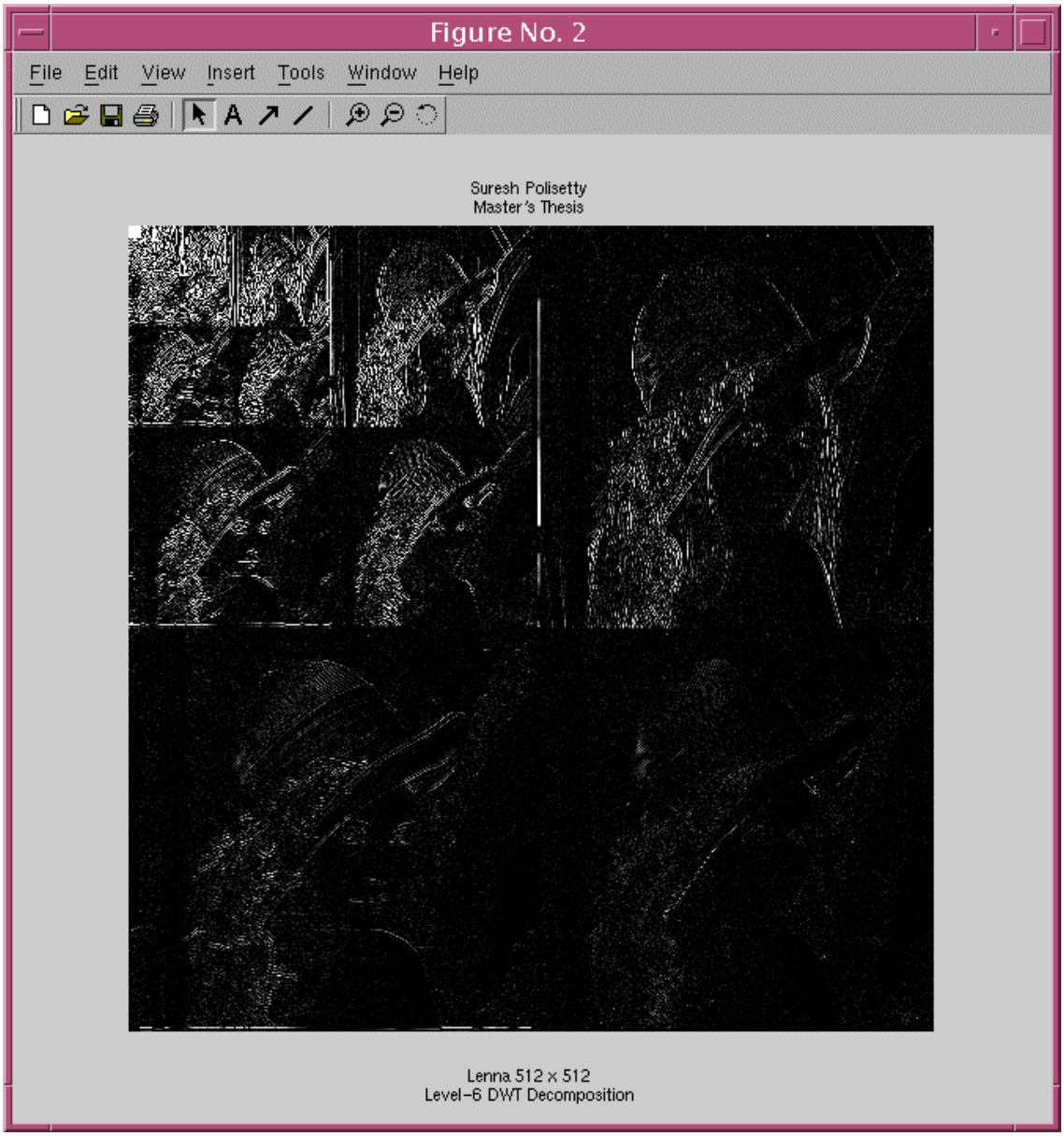


Figure 43 Lena 512 x 512, 6-scale DWT Decomposition

The compressed images can be seen from Figure 44 to 51. The reconstructed image using all the 42848 bytes is shown in Figure 52. The compression ratios and PSNR (Peak-to-Signal Noise Ratio) values are tabulated and can be seen in Table 9.

4.1.2 Test Image – Barbara

The original Barbara image in PGM format is shown in Figure 53. The wavelet-decomposed image is shown in Figure 54. The Encoder has generated a bitstream file containing 45504 bytes, or at a bit rate of 0.1736 bpp (bits per pixel). Here also, various images were reconstructed using an assortment of number of bytes. At a compression ratio 1024 to 1, the image "Barbara" also was not recognizable. However, at the compression ratio of 512 to 1, it could be recognized even though the image quality was poor.

Table 9 Experimental results of Lena

Number of Bytes used	Bpp (bits per pixel)	Compression Ratio	PSNR
32768	1.0	8:1	15.7464
16384	0.5	16:1	15.7446
8192	0.25	32:1	15.6687
4096	0.125	64:1	15.5962
2048	0.0625	128:1	15.5508
1024	0.03125	256:1	15.5293
512	0.015625	512:1	15.4013
256	0.0078125	1024:1	15.3440



Figure 44 L-Compr. Ratio 1024 : 1



Figure 45 L-Compr. Ratio 512 : 1



Figure 46 L-Compr. Ratio 256 : 1



Figure 47 L-Compr. Ratio 128 : 1



Figure 48 L-Compr. Ratio 64 : 1



Figure 49 L-Compr. Ratio 32 : 1



Figure 50 L-Compr. Ratio 16 : 1



Figure 51 L-Compr. Ratio 8 : 1

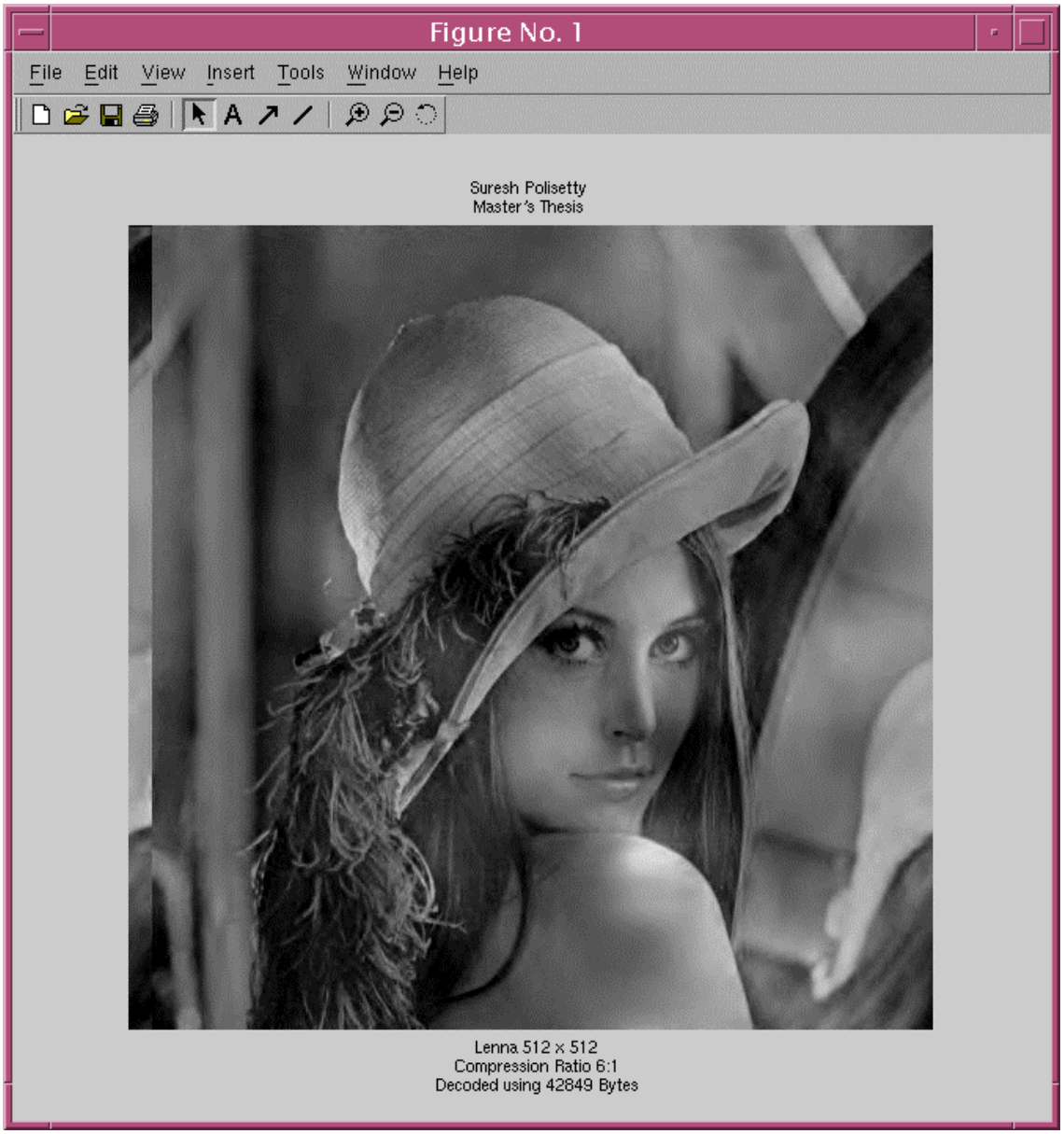


Figure 52 Reconstructed Lena Image using all 42848 bytes

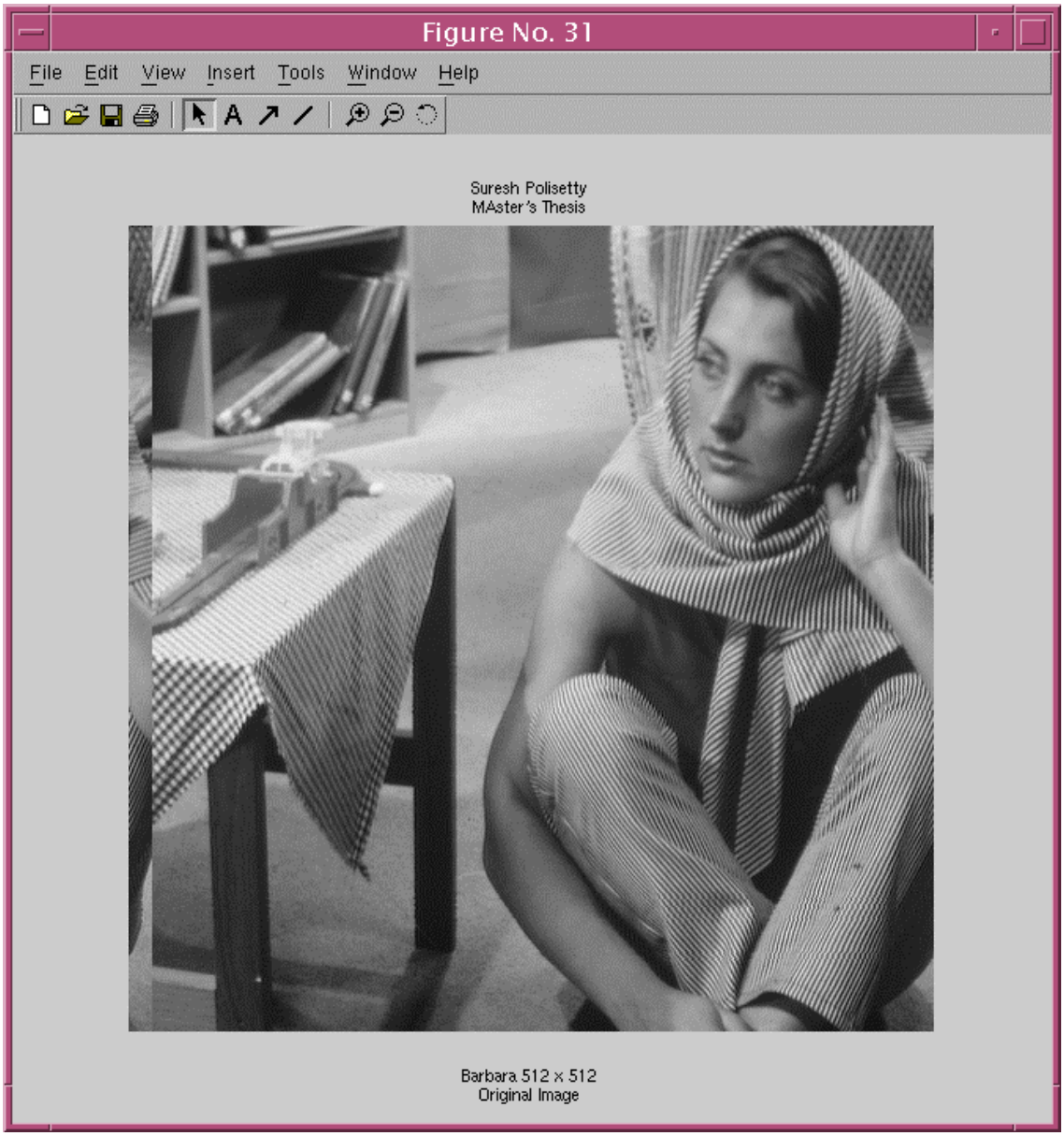


Figure 53 Test Image – Lena 512 x 512 Original

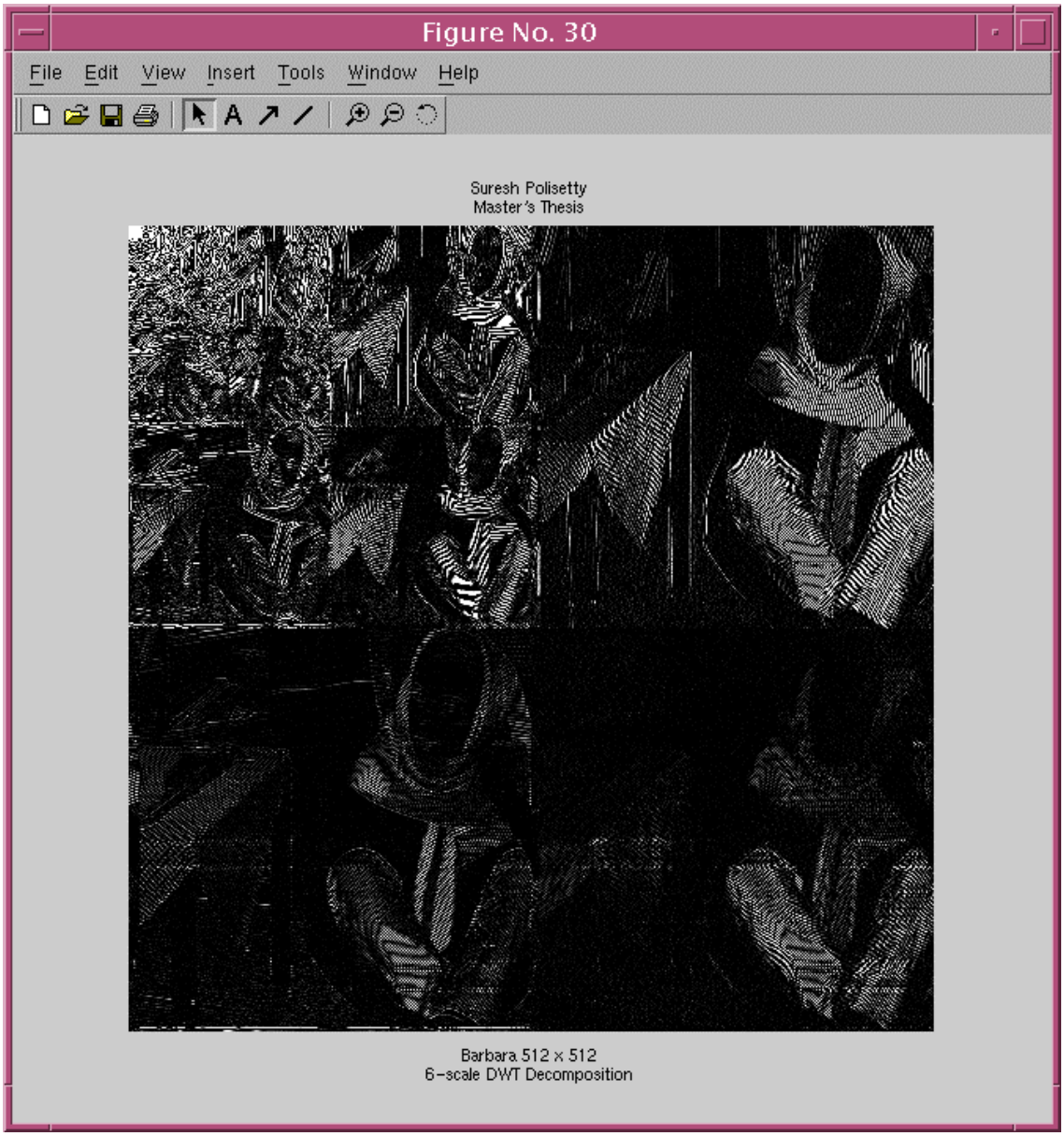


Figure 54 Barbara 512 x 512, 6-scale DWT Decomposition

Table 10 Experimental results of Barbara

Number of Bytes used	Bpp (bits per pixel)	Compression Ratio	PSNR
32768	1.0	8:1	16.6070
16384	0.5	16:1	16.4393
8192	0.25	32:1	16.1832
4096	0.125	64:1	15.9655
2048	0.0625	128:1	15.8896
1024	0.03125	256:1	15.7872
512	0.015625	512:1	15.6036
256	0.0078125	1024:1	15.2585

The compressed images can be seen from Figure 55 to 62. The reconstructed Image using all the 45504 bytes is shown in Figure 63. The compression ratios and PSNR (Peak-to-Signal Noise Ratio) values are tabulated and can be seen in Table 10.

4.1.3 Test Image – Goldhill

The original Goldhill image in PGM format is shown in Figure 64. The wavelet-decomposed image is shown in Figure 65. The Encoder has generated a bitstream file containing 45528 bytes, or at a bit rate of 0.1737 bpp (bits per pixel). Here also, various images were reconstructed using an assortment of number of bytes. At a compression ratio 1024 to 1, the image, "Goldhill," also was not recognizable. Even at the compression ratio of 512 to 1, it was not recognizable. This might be due to the broken edge information and also the other two images were human faces in close-up shot. However, at the compression ratio of 256 to 1, it could be recognized even



Figure 55 B-Compr. Ratio 1024 : 1



Figure 56 B-Compr. Ratio 512 : 1



Figure 57 B-Compr. Ratio 256 : 1



Figure 58 B-Compr. Ratio 128 : 1



Figure 59 B-Compr. Ratio 64 : 1



Figure 60 B-Compr. Ratio 32 : 1



Figure 61 B-Compr. Ratio 16 : 1



Figure 62 B-Compr. Ratio 8 : 1

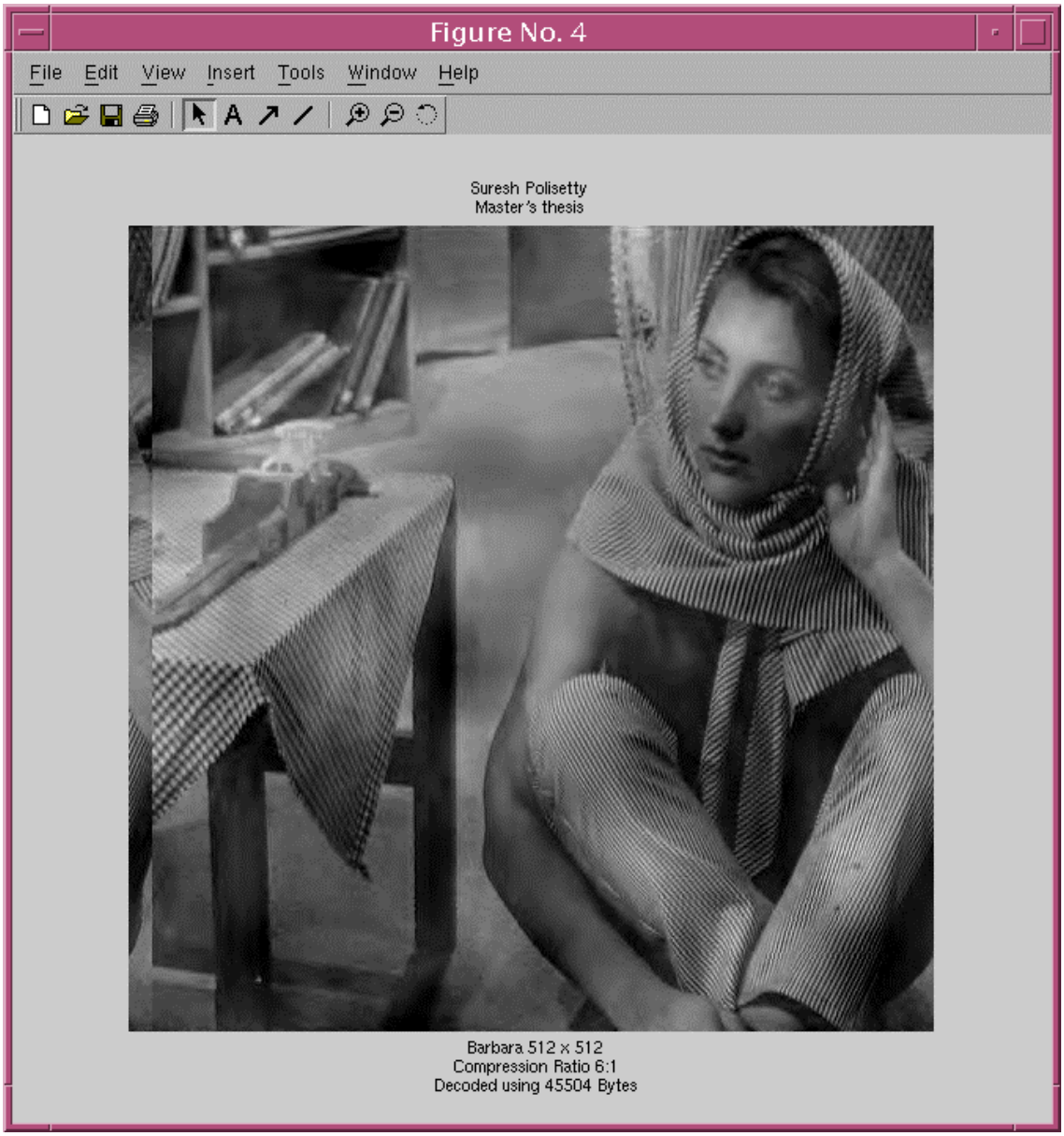


Figure 63 Reconstructed Barbara Image using all 45504 bytes

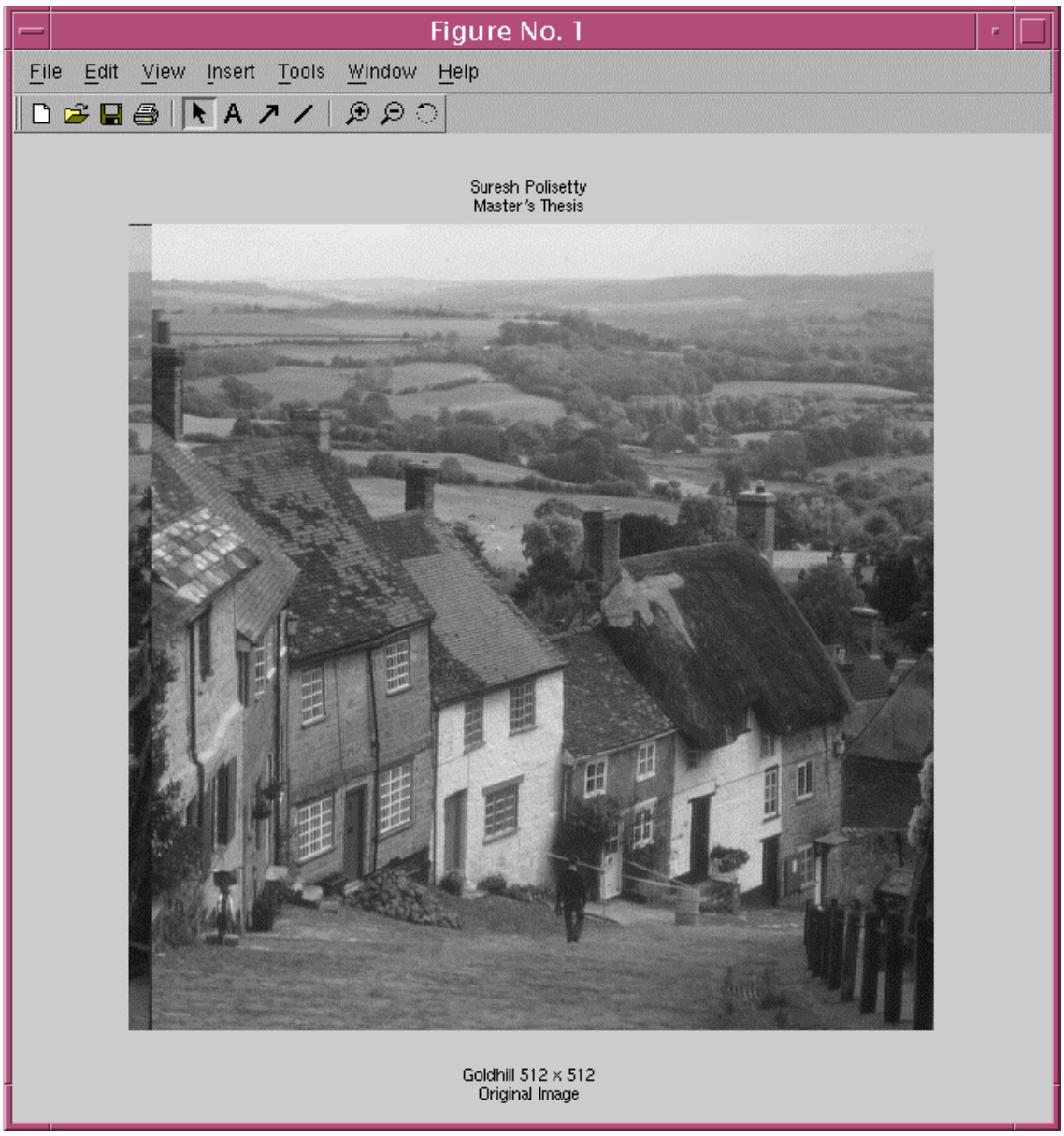


Figure 64 Test Image – Goldhill 512 x 512 Original

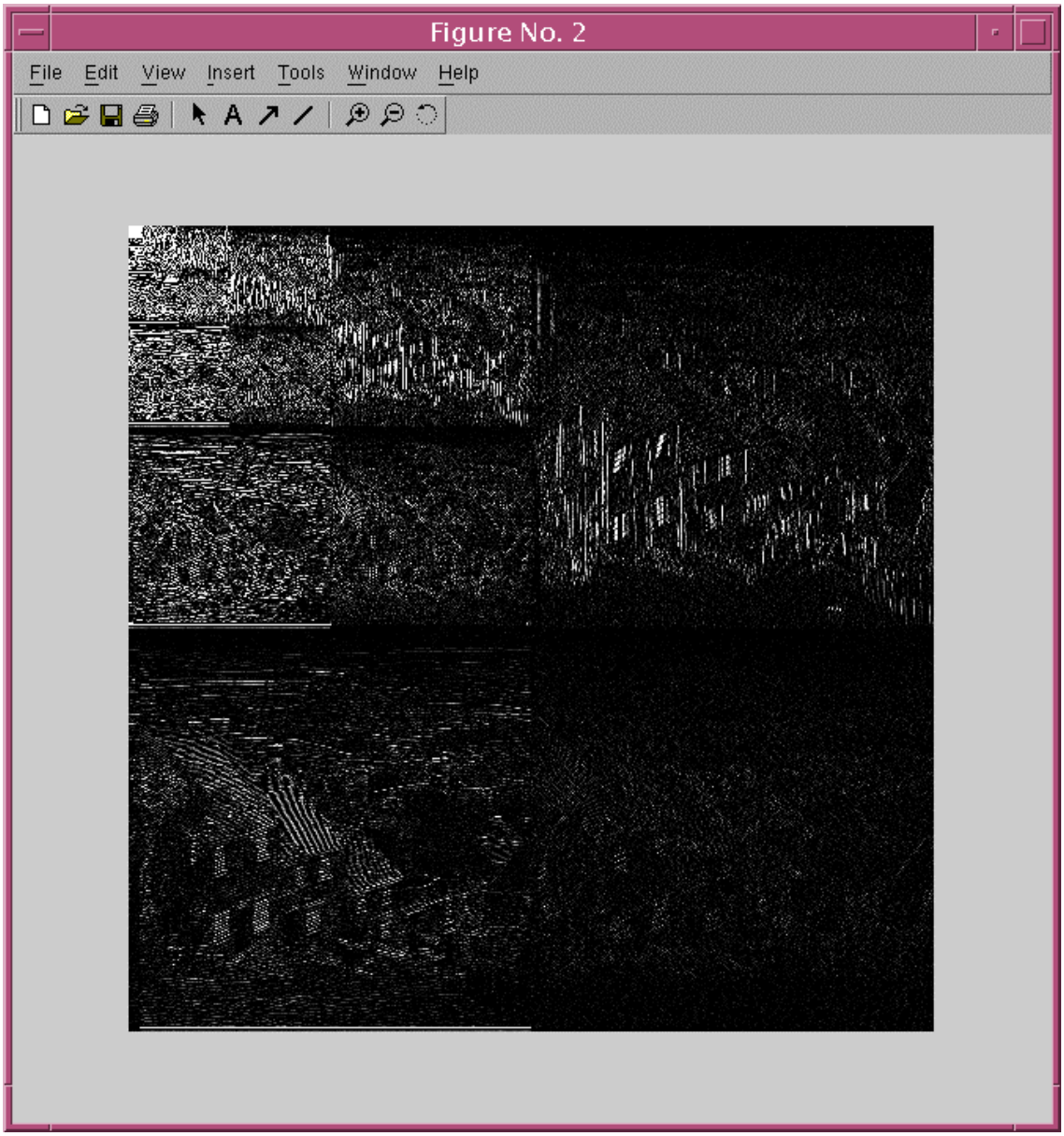


Figure 65 Goldhill 512 x 512, 6-scale DWT Decomposition

though the image quality was poor. The compressed images can be seen from Figure 66 to 73. The reconstructed image using all the 45528 bytes is shown in Figure 74. The compression ratios and PSNR (Peak-to-Signal Noise Ratio) values are tabulated and can be seen in Table 11.

4.2 Hardware

The results obtained from software and hardware implementations are found to be identical. The flow explained in the Figure 75 is used for the test Images, Lena, Barbara and Goldhill. This part of the work was fully implemented using MATLAB to be certain the EZW algorithm was fully understood and to serve as a validation reference. The validation of the MATLAB code is done when the reconstructed images are found to be visually matching to the original images. The flow explained in the Figure 76 is used for the hardware implementation of the EZW algorithm on an 8x8 image, which was given as an example in [1].

Table 11 Experimental results of Goldhill

Number of Bytes used	Bpp (bits per pixel)	Compression Ratio	PSNR
32768	1.0	8:1	16.5982
16384	0.5	16:1	16.5625
8192	0.25	32:1	16.4782
4096	0.125	64:1	16.3260
2048	0.0625	128:1	16.2759
1024	0.03125	256:1	16.2174
512	0.015625	512:1	15.8473
256	0.0078125	1024:1	15.6303



Figure 66 G-Compr. Ratio 1024 : 1



Figure 67 G-Compr. Ratio 512 : 1



Figure 68 G-Compr. Ratio 256 : 1



Figure 69 G-Compr. Ratio 128 : 1



Figure 70 G-Compr. Ratio 64 : 1



Figure 71 G-Compr. Ratio 32 : 1



Figure 72 G-Compr. Ratio 16 : 1



Figure 73 G-Compr. Ratio 8 : 1

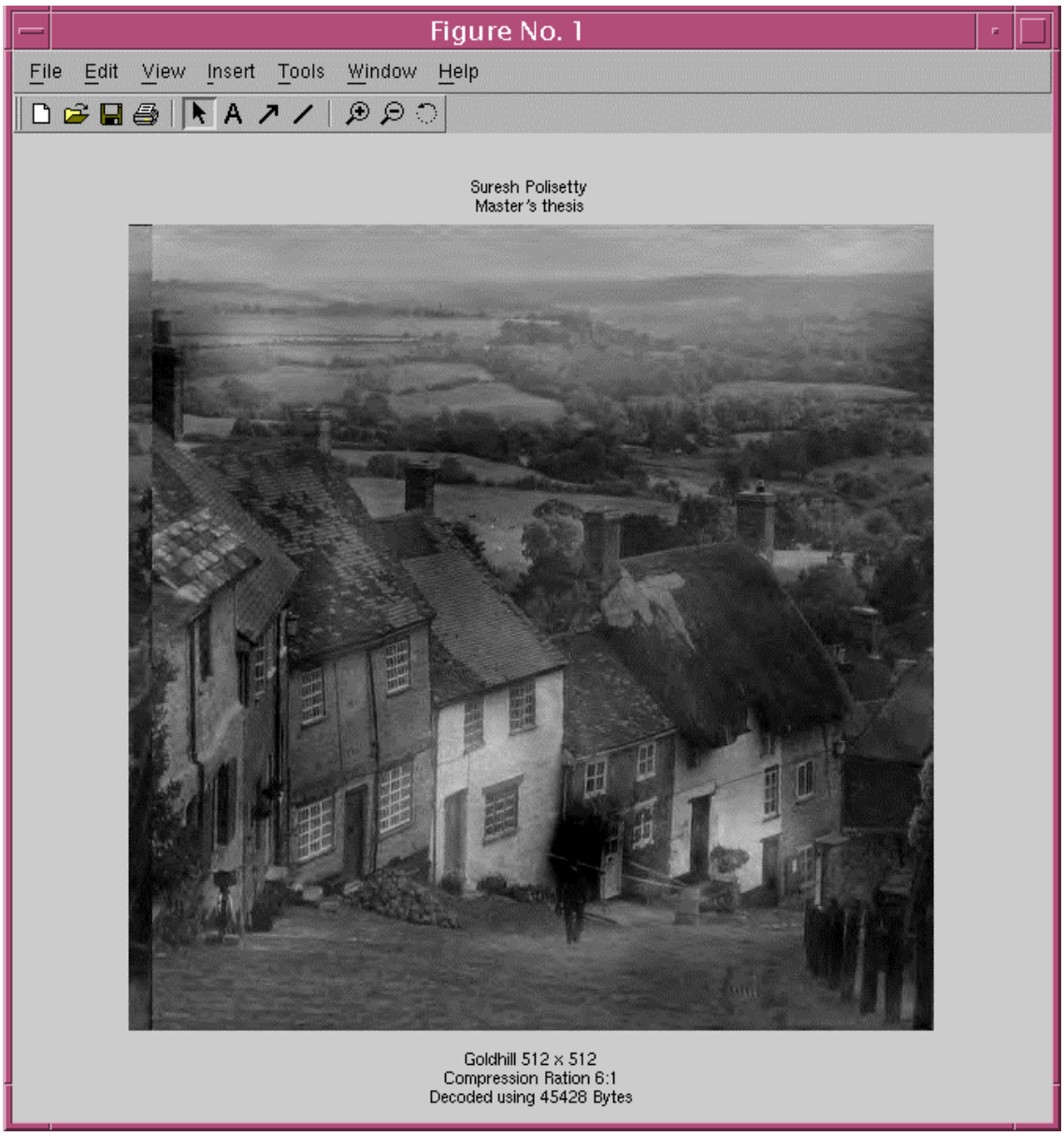


Figure 74 Reconstructed Goldhill Image using all 45428 bytes

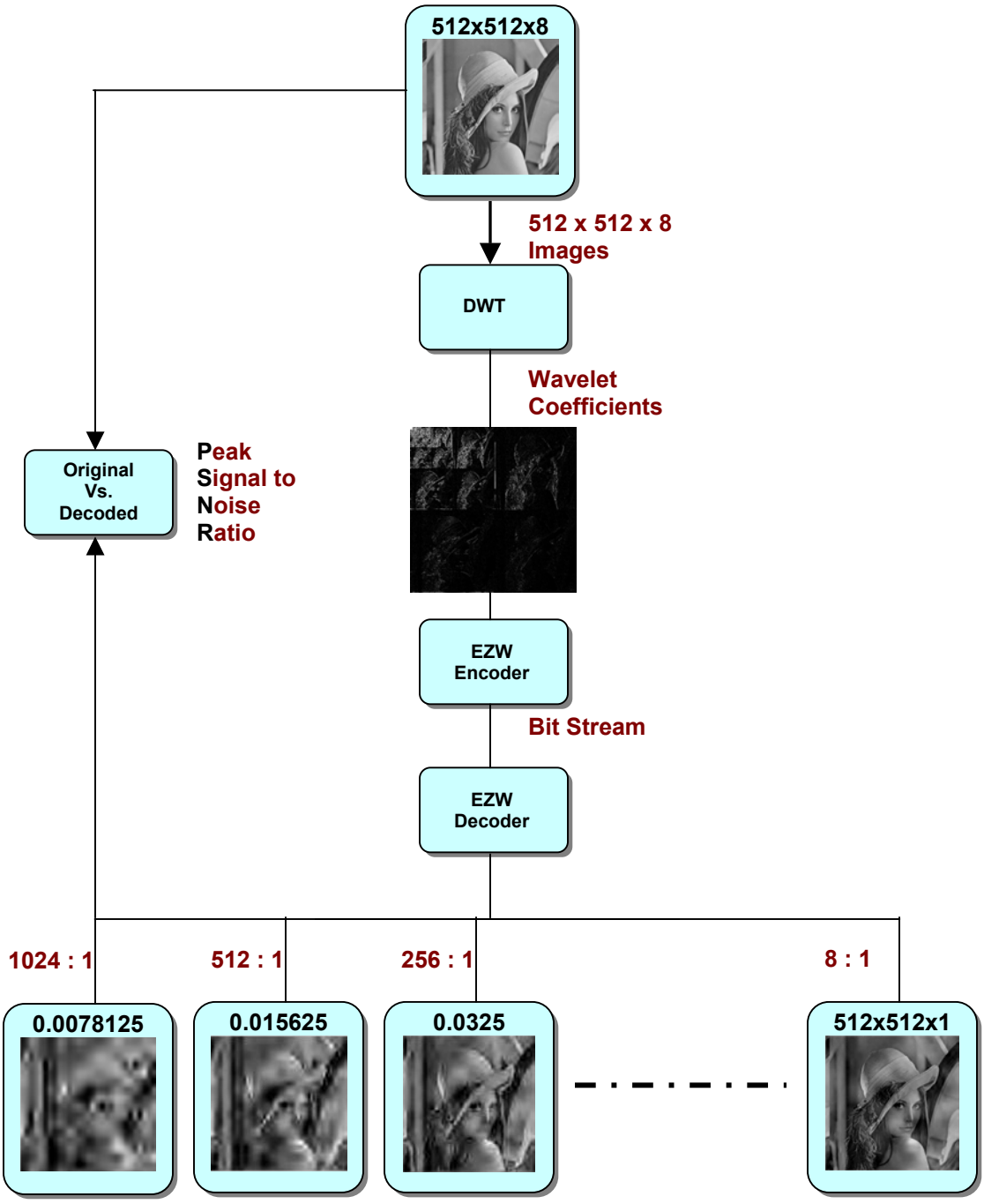


Figure 75 MATLAB Implementation for Multiple Compression Ratios

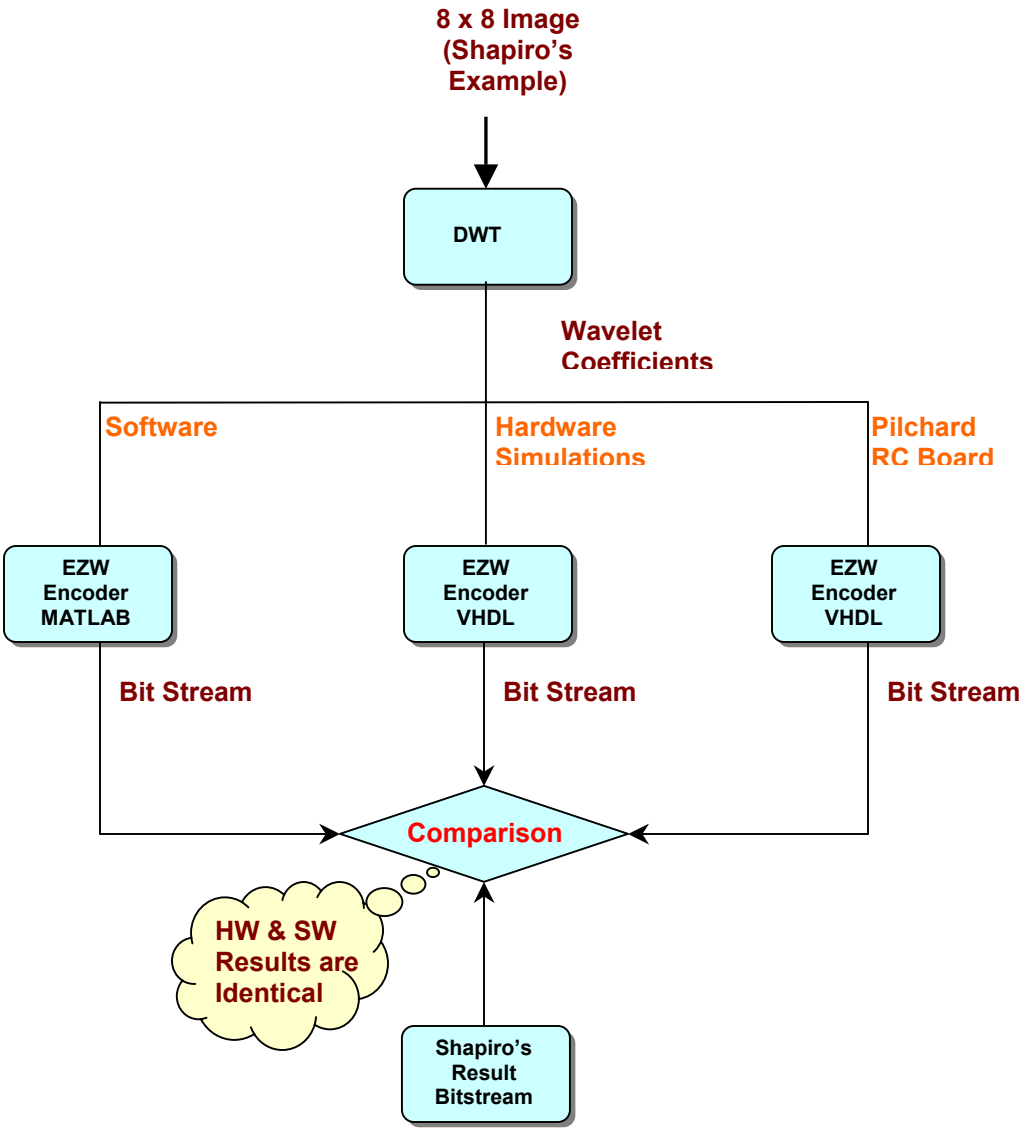


Figure 76 Verification of the Hardware Implementation

4.2.1 Area and Speed

The EZW encoder occupies about 81 % of the Virtex™ 1000E FPGA. The circuit can operate at a maximum speed of 11.5 MHz. The design was tested on the Pilchard board at a speed of 8.315 MHz.

4.2.2 Limitations

The hardware implementation of the EZW encoder on Pilchard RC platform has certain limitations. The following subsections can explain the details.

4.2.2.1 Limitaion#1

At the beginning of the design while setting up the specifications, the hurdle comes in the form of memory, as the EZW algorithm requires the whole image at a time for encoding. The typical Dual-Port RAM used for the Pilchard board was limited to a maximum of 256 memory-lines. This is due to the fact that, on Pilchard, only 8 address bits are available. The maximum width of each row is limited to 64 bits because of the 64-bit I/O Data-BUS interfacing. Unlike the regular image, the representation of the pixel values of the input image containing wavelet coefficients needs more than 8 bits. It is expected that, including the extra bit for denoting the sign, each pixel can be represented with 16 bits. Thus, the maximum size of the image that can be encoded at one time is restricted to 32x32, since the number 32x32x16 is equal to 256x64.

Table 12 Area and Speed

Size of the Image	Maximum Speed	Area	Success/Failure
32x32	Failed at Synthesis	-	Failure
16x16	Failed at Synthesis	-	Failure
8x8	11.5 MHz	81%	Success

4.2.2.2 Limitaion#2

After the design was simulated successfully, it was then arranged for synthesis for a 32x32 image. However, the synthesis failed as the design was too big for the particular target FPGA, which is Xilinx's Virtex™ 1000E. Then the next possible image size that is 16x16 was arranged for synthesis and the result was also failure. Finally, the design was set up for synthesis for an 8x8 block. This time the outcome was successful. When placed and routed, the design used 81% of the resources of the Xilinx's Virtex™ 1000E FPGA. The consequences were tabulated and can be seen in Table 12.

4.3 Hardware Vs. Software

For an image size of 8x8, the software implementation took approximately 1 second. For the same image, the time taken for the hardware implementation was calculated using the results obtained. The total time taken by the simulations was noted as 317400 ns, whereas the "clk" and "clk_div" frequencies were

set up at 5 MHz and 2.5 MHz during the simulations. The "clk" denotes the system clock and the "clk_div" denotes the reduced clock rate. Thus, the total number of clock cycles can be calculated as below.

1 clock cycle (clk_div of Modelsim) = 1 / (2.5 MHz) => 400 ns.

Total number of clock cycles for EZW Encoder = 317400 / 400 => 793.5

The Virtex EZW Encoder uses a clock rate of 8.315 MHz. Thus,

1 clock cycle (clk_div of Pilchard) = 1 / (8.315 MHz) = 120.26 ns.

The expected total time could be taken by the EZW Encoder for an image size of 8 x 8 is calculated as below.

$T_{Software}$ = 1×10^9 ns

$T_{Hardware}$ = Total Number of Clock Cycles x Clock Delay

= 793.5 x 120.26 ns

= 95,426.31 ns

Speed-up = $T_{Software} / T_{Hardware}$

= 1×10^9 ns / 95426.31 ns

= 10,479

4.4 Speedup and Desired Architecture

The following subsections are intended to discuss the data transfers involved in the hardware/software implementations and also the other possibilities to achieve speedup.

4.4.1 Speedup Including the Time for Data Transfer

The speedup achieved through hardware was previously calculated without considering the time taken for the data transfer between the Pentium III and the Virtex 1000e on the Pilchard. Also, the total time taken by MATLAB to load data on the SPARC 280 was included in the previous speedup calculations. Thus, new speed-ups achieved were calculated with the new times, including and excluding data transfer times both in the hardware and software implementations.

Case 1: Excluding data transfer

The time taken for MATLAB to read the inputs and to write the outputs with no operations involved was observed to be approximately 300 milliseconds. This time of data transfer was subtracted from the total time, previously noted as approximately one second. However, the time estimated previously on the Virtex hardware was only for the operations, not for the data handling. Therefore, the new speedup excluding data transfer is calculated as below:

$T_{Software} = (1\text{-}0.3) \times 10^9 \text{ ns} = 7 \times 10^8 \text{ ns}$

$T_{Hardware} = 95{,}426.31 \text{ ns}$

$\text{Speed-up} = T_{Software} / T_{Hardware}$

$= 7 \times 10^8 \text{ ns} / 95426.31 \text{ ns}$

$= 7{,}335$

Case 2: Including data transfer

In this case, the time taken for the MATLAB to read the inputs and to write the outputs is included in the software implementation time. Also, the time taken for the C routine to read and write the inputs and outputs (time for data transfer between

the Pentium III and the Virtex 1000e on the Pilchard) from the FPGA was included in the hardware implementation time. Therefore, the new speedup including data transfer is calculated to be:

$T_{Software}$ = 1×10^9 ns

$T_{Hardware}$ = 95426.31+50000 ns

= 145,426.31

Speed-up = $T_{Software} / T_{Hardware}$

= $1x\ 10^9$ ns / 145426.31 ns

= 6,876

It is obvious that the speedup achievable is decreased with the inclusion of data transfer. However, the speedup of 6,876 is still substantial.

4.4.2 Other Possibilities to Achieve Speedup

If the MATLAB functions were converted to C routines, then the software speedup could be at least 10x. This conversion can be done with the aid of software automatic converters. For example, the MATLAB descriptions could be converted to FORTRAN using the converter *matlab2fmex* and the resulting FORTRAN could then be converted to the C descriptions using a converter such as *F2C*.

When the EZW algorithm is implemented on an ASIC with an internal RAM, even better performance than the FPGA using DIMM interfacing can be achieved due to the faster clock frequency possible with the ASIC. Thus, the VHDL model of the EZW algorithm was implemented on an ASIC targeting the TSMC-0.18 process. Though the post-layout simulations were not conducted due to personal time

constraints, interesting results were obtained. The ASIC containing the EZW and the required RAMs could resemble Figure 77, which shows a system-on-chip developed by other graduate students in our UTK Microelectronic Systems Laboratory [26]. The number of transistors was determined from the resulting netlist of the EZW design. The EZW design which can handle an 8 x 8 image requires 157,419 transistors, not counting the RAM. If the design were scaled for an image of size 512 x 512 (4096 times larger than the 8 x 8), the total number of transistors would be 644,788,224 (almost 650 million) transistors, again not including the RAM. This shows the fact that the EZW algorithm in its present formulation is not amenable to a cost-effective hardware implementation.

The EZW hardware implementation results are compared below with the results for the DCT, which was implemented on hardware by previous graduate students in our Laboratory [27]. The DCT algorithm was formulated to handle 8 x 8 blocks of an image. The DCT hardware design is insensitive to the size of the image so it can be applied to any image irrespective of the size. When targeted to the TSMC-0.18 process, the DCT design required 33,112 transistors. From these results, it is noticeable that the hardware implementation of the DCT is much more cost-effective than the EZW. Table 13 gives a brief summary of the comparison between the ASIC feasibility of the DCT and the EZW. The results presented above were tabulated and can be seen in Table 14.

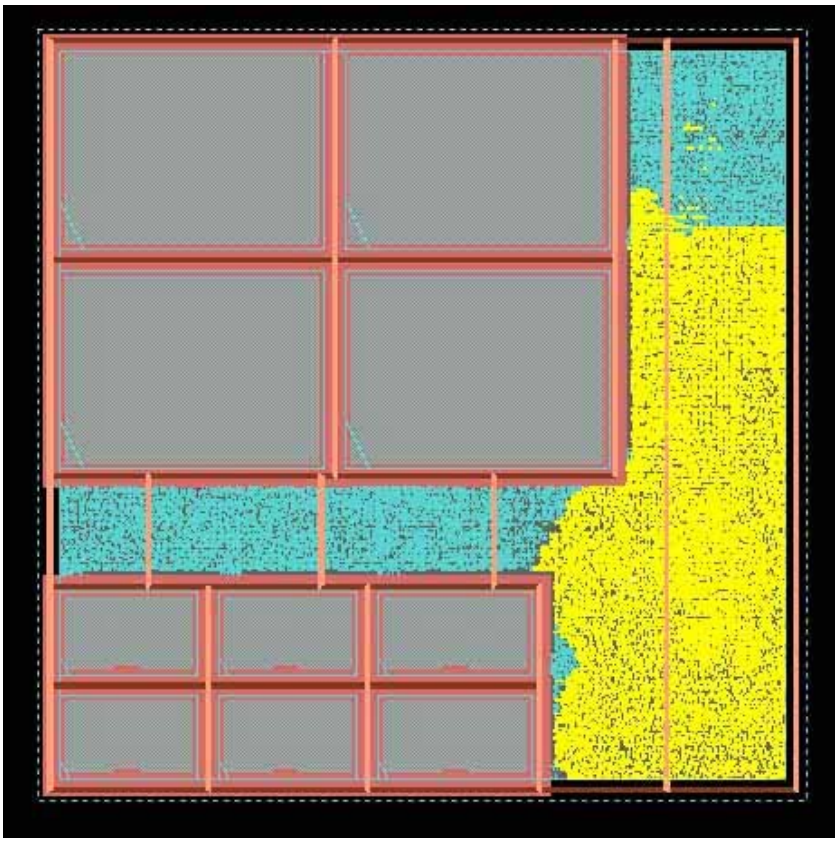


Figure 77 An example system-on-chip platform [26]

Table 13 Comparison between DCT and EZW on ASIC.

Algorithm		Number of Transistors	Comments
DCT	for 8x8 images	33,112	Could be applied to any image irrespective of the size
	for 512x512 images	33,112 (1x)	
EZW	for 8x8 images	157,419	Exclusively designed to target images of particular size.
	for 512x512 images	644,788,224 (4096x)	

Table 14 Possible EZW Speed-ups on Different Platforms

Platform	Specifications	Speed-up	Comments
MATLAB	Couple of hundreds lines of Code	1x	Easy to implement. DWT and IDWT (inverse DWT) can be performed using MATLAB built-in functions.
C	Couple of hundreds lines of Code but complexity level is higher compared to the MATLAB Code	10x	DWT and IDWT have to be implemented before EZW Encoder and after EZW Decoder respectively. Alternatively, the wavelet transformations can be performed separately using MATLAB so that input to the C platform are the wavelet coefficients and the output is a stream of binary bits.
FPGA	81% or 9955/12288 Slices of the Xilinx Virtex 1000e. Memory interfacing unit.	6876x	Maximum speed achieved was 11.5 MHz and the design was tested at a clock frequency of 8.315 MHz on Pilchard Reconfigurable Computing System.
ASIC	157,419 transistors, not counting an internal RAM.	20,628x to 41,256x	If the design works at a clock frequency of 25MHz to 50MHz then 3 times the FPGA to 6 times the FPGA speed-up can be achieved.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

A VHDL model for the Embedded Zerotree Wavelet algorithm has been developed. Significant acceleration was achieved since the hardware implementation in a FPGA (Xilinx Virtex-1000E using a 8.315 MHz clock) ran 10,000 times faster than the MATLAB implementation on a SUN-220 workstation. Additional speedup exploiting the parallel capabilities of the FPGA was not achieved since the EZW algorithm utilizes only sequential operations.

5.2 Future work

What has been discussed and implemented in this work is just an initiation for hardware acceleration of the Embedded Zerotree Wavelet algorithm. The design is confined to only 8x8 size images due to the fact that EZW algorithm requires whole image while encoding and also due to the hardware limitations. The design could be targetted to a bigger FPGA with enough memory to implement images of different sizes or an application-specific integrated circuit (ASIC) with a RAM capable of holding huge images. Additional speedup can be possible if the EZW algorithm is formulated in a manner that some operations can be done parallel.

REFERENCES

1. J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing,* vol. 41, Dec. 1993, pp. 3445–3462.

2. Charles D. Creusere, "A New Method of Robust Image Compression Based on the Embedded Zerotree Wavelet Algorithm." pp. 1436-1441

3. J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients", *IEEE Trans. Signal Processing*, December 1993 . [Online] available: http://www.ws.binghamton.edu/fowler/fowler personal page/EE523_files/Embedded Image Coding Using Zerotrees.pdf

4. Image compression using transformations, ELEC 539 PROJECT REPORT. [Online] available: http://www.owlnet.rice.edu/~elec539/Projects99/BACH/proj1/report/index.html

5. "Zerotree Wavelet Using Fractal Prediction." [Online] available: http://www.f4.fhtw-berlin.de/~barthel/paper/BBHH97.pdf

6. "An Improved Embedded Zerotree Wavelet Image Coding Method Based On Coefficient Partitioning using morphological Operation." [Online] available: http://www.worldscinet.com/ijprai/14/preserveddocs/1406/S0218001400000490.pdf

7. Image Compression - from DCT to Wavelets: A Review by Subhasis Saha. [Online] available: http://www.acm.org/crossroads/xrds6-3/sahaimgcoding.html

8. Taekon Kim, Robert E. Van Dyck, and David J. Miller. "Hybrid Fractal Zerotree Wavelet Image Coding." [Online] available: http://w3.antd.nist.gov/pubs/fzw_2002.pdf

9. Jie Liang. Highly Scalable Image Coding for Multimedia Applications, [Online] available: http://www.acm.org/sigs/sigmm/MM97/papers/liang/acm97.html

10. Iraj Sodagar, Hung-Ju Lee, Paul Hatrack, and Ya-Qin Zhang. "Scalable Wavelet Coding for Synthetic/Natural Hybrid Images." [Online] available: http://research.microsoft.com/china/papers/Scalable_Wavelet_Coding_Synthetic_Images.pdf

11. Mike Goldsmith, VHDL Tutorial [Online] available: http://www.asic.uwaterloo.ca/groups/digital/mgoldsmith/VHDL_Tutorial_1.pdf

12. 4. P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and K. H. Lee, "Pilchard - A Reconfigurable Computing Platform With Memory Slot Interface", *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2001.

13. "Frequently Asked Questions and Answers on Virtex-E." [Online] available: http://www.xilinx.com/prs_rls/vtxefaq.htm

14. JPEG2000 Versus JPEG "Classic". [Online] available: http://www.microimages.com/documentation/cplates/67jpeg2000versus.pdf

15. Marcin Kociołek1, Andrzej Materka1, Michał Strzelecki1, Piotr Szczypiński, "Discrete Wavelet transform – Derived features for digital Image texture Analysis." [Online] available: http://www.eletel.p.lodz.pl/cost/pdf_9.pdf

16. C. Valens, "Embedded Zerotree Wavelet Encoding." [Online] available: http://perso.wanadoo.fr/polyvalens/clemens/ezw/ezw.html

17. Xilinx Inc., [Online] available: http://www.xilinx.com/

18. "VHDL Tutorial," [Online] available: http://www.vhdlonline.de/tutorial/englisch/t_219.htm

19. Eui-Sung Kang1, Toshihisa Tanaka2, Tae-Hyung Lee1, and Sung-Jea Ko1, "A Multi-threshold Embedded Zerotree Wavelet Coder." [Online] available: http://sip-www.ei.tuat.ac.jp/~tanaka/publications/mapaper.pdf

20. Taekon Kim, Member, IEEE, Seungkeun Choi, Robert E. Van Dyck, Member, IEEE, and Nirmal K. Bose, Fellow, IEEE, "Classified Zerotree Wavelet Image Coding andAdaptive Packetization for Low-Bit-Rate Transport." [Online] available: http://w3.antd.nist.gov/pubs/vandyck01.pdf

21. Jon K. Rogers and Pamela C. Cosman, Member, IEEE, "Wavelet Zerotree ImageCompression with Packetization." [Online] available: http://code.ucsd.edu/~pcosman/web-11.pdf

22. S. Areepongsa, N. Kaewkamnerd, Y. F. Syed and K. R. Rao, "Wavelet Based Compression for Image Retrieval Systems." [Online] available: http://www-ee.uta.edu/dip/paper/CHC-RIOT_11.PDF

23. Xiaoyan Xu, "Embedded Zero Tree as Image Coding." [Online] available: http://www.uoguelph.ca/~xux/courses/ENGG6560.pdf

24. Zixiang Xiong, Kannan Ramchandran, Michael T. Orchard, and Ya-Qin Zhang, "A Comparative Study of DCT- and Wavelet-Based Image Coding." [Online] available: http://research.microsoft.com/china/papers/Comparative_Study_DCT_WaveletBased_Image_Coding.pdf

25. N. J. Mitra, P. K. Biswas, T. Acharya, “Modified Embedded Zerotree Scheme for Efficient Coding of Discrete Wavelet Coded Frames.” [Online] available: http://www.iiit.net/research/cvit/icvgip00/I-56.pdf

26. R. Srivastava, ``Development of an Open Core System-on-Chip Platform’’, M.S. Thesis, University of Tennessee, August 2004. [Online] available: http://vlsi1.engr.utk.edu/ece/rishi-thesis.pdf

27. Gabi Chereches, Kamesh Ramani, Madhan, Mardav Wala, “Discrete Cosine Transform”, ECE552 Course Project, University of Tennessee, May 2003. [Online] available: http://vlsi1.engr.utk.edu/~gabi/552/dct/report/home.html

VITA

Suresh Polisetty was born in Tallapuram, India. He grew up and did his schooling in Kakinada, India. He went to P.R.G Jr College, Kakinada, for his post-school education. He then went to the J.N.T.U College of Engineering, Kakinada, and obtained his Bachelor of Technology degree in Electrical and Electronics Engineering in 2001. He joined the University of Tennessee, Knoxville to pursue his graduate studies. Subsequently he has been doing his research under the guidance of Prof. Donald W. Bouldin. He plans to graduate with a Master's degree in Electrical Engineering in December 2004.