2004

Wavelet Transform Based Image Compression on FPGA

Faizal Iqbal

FLORIDA STATE UNIVERSITY

COLLEGE OF ENGINEERING

WAVELET TRANSFORM BASED IMAGE COMPRESSION ON FPGA

By

FAIZAL IQBAL

A Thesis submitted to the
Department of Electrical and Computer Engineering
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 2004

The members of the Committee approve the thesis of Faizal Iqbal defended on 27th of February 2004.

Simon Y Foo
Professor Directing Thesis

Uwe Meyer-Baese
Committee Member

Rodney Roberts
Committee Member

Approved:

Reginald Perry, Chair, Department of Electrical and Computer Engineering

Ching-Jen Chen, Dean, FAMU-FSU College of Engineering

The Office of Graduate Studies has verified and approved the above named committee members.

Dedicated to my family

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Simon Foo for his guidance and support throughout my graduate study at FSU. I would also like to thank the members of my thesis committee, Dr. Uwe Meyer-Baese and Dr. Rodney Roberts, for their valuable advice and guidance. I wish to thank the academic and administrative staff at the Department of Electrical and Computer Engineering for their kind support. I wish to thank my family for their continuous support and confidence in me. I also wish to thank my friends for their support.

TABLE OF CONTENTS

LIST OF ACRONYMS .... **viii**

LIST OF TABLES .... **x**

LIST OF FIGURES .... **xi**

ABSTRACT .... **xiii**

CHAPTER 1 INTRODUCTION **1**

1.1 Motivation for Research 1

1.2 Background 1

1.3 Organization of Thesis 3

CHAPTER 2 WAVELET TRANSFORM BASED IMAGE COMPRESSION **4**

2.1 Introduction 4

2.2 Principles of Image Compression 5

2.3 Wavelet Transform as the Source Encoder 6

2.3.1 Measuring Frequency Content by Wavelet Transform 7
2.3.2 Wavelet-based Compression 7

2.4 Wavelet Decomposition 8

2.5 Embedded Zero Tree Encoding Technique. 11

CHAPTER 3 FPGA TOOLS & METHODS **14**

3.1 FPGA Programmable Logic 14

3.1.1 FPGA Benefits 15
3.1.2 SRAM-based FPGA 15

3.2 HDL-VHDL 17

3.2.1 Entities 18
3.2.2 Architectures 19

3.2.3 Configurations 19
3.2.4 Generic Parameters 20

3.3 XESS XSV Virtex FPGA Board 21

3.4 Software Tools 23

CHAPTER 4 SOFTWARE IMPLEMENTATION 25

4.1 1-D Wavelet Transform 25

4.2 Implementation in Multi Dimension 27

4.3 Daubechies Wavelet Implementation 30

4.4 Software Implementation Results 33

4.4.1 Biorthogonal Cohen-Daubechies–Feuveau Wavelet 33
4.4.2 Daubechies Wavelet 33

CHAPTER 5 HARDWARE IMPLEMENTATION USING FPGAS 34

5.1 Implementation Overview 34

5.2 Implementation Scheme for DWT 37

5.3 VHDL Modules 39

5.4 MATLAB Modules 41

CHAPTER 6 HARDWARE IMPLEMENTATION- RESULTS AND ANALYSIS 42

6.1 Simulation & Results 42

6.2 Test Results 44

6.3 Performance (Area & Speed) 47

CHAPTER 7 CONCLUSIONS AND FUTURE EXTENSIONS 49

7.1 Conclusions 49

7.2 Future Work 50

APPENDIX A 52

APPENDIX B 62

APPENDIX C **65**

APPENDIX D **72**

REFERENCES **73**

BIOGRAPHICAL SKETCH **74**

LIST OF ACRONYMS

1D Short form for one dimensional.
2D Short form for two dimensional.
CLB A *Configurable Logic Block* is a physically co-located grouping of LUT's, flip flops, and carry logic.
CPU A *CPU* is the part of a computer that interprets and executes instructions.
DB4 Short form for Daubechies wavelet filter with 4 taps.
DCT Discrete Cosine Transform.
DFT Discrete Fourier Transform.
DWT Discrete Wavelet Transform.
DLL The *Delay-Locked Loops* can be used to eliminate skew between the clock input pad and the internal clock input pins throughout the device. Moreover, the DLL provides advance control of multiple clock domains and may operates as a clock mirror.
EEPROM An *Electrically Erasable PROM* is a non-volatile storage device using a technique similar to the floating gates in EPROMs but with the capability to discharge the floating gate electrically. Usually bytes or words can be erased and reprogrammed individually during system operation.
EPROM An *Erasable-Programmable Read-Only Memory* is a PROM that can be erased by exposure to ultraviolet light and then reprogrammed.
EZT Embedded Zero Tree (an image encoding technique).
FPGA A *Field-Programmable Gate Array* is a type of logic chip that can be programmed. An FPGA is similar to a PLD, but whereas PLDs are generally limited to hundreds of gates, FPGAs support thousands of gates. They are especially popular for prototyping integrated circuit designs. Once the design is set, hardwired chips are produced for faster performance.
FSM Finite State Machine (Refers here to a coding technique in HDL).
HDL *Hardware Description Language* is a kind of language used for the conceptual design of integrated circuits. Examples are VHDL and Verilog.
HVS Human Visual System.
GCC GNU Compiler Collection (An open source C compiler).
IC An *Integrated Circuit* is a tiny slice or chip of material on which is etched or imprinted a complex of electronic components and their interconnections.
IEEE The *Institute of Electrical and Electronics Engineers*, founded in 1884, is an organization composed of engineers, scientists, and students. The IEEE is best know for developing standards for the computer and electronics industry.
IOB a physically co-located group of buffers, latches, flip flops, and input- /output pads used for sending signals off of the FPGA and receiving signals onto the FPGA.
JPEG Joint Photographic Experts Group (an image compression format). JPEG uses an 8x8 grid and does a discrete cosine transformation on the image.

JPEG2000 A newer more computationally intensive JPEG standard. It allows for much higher compression rates than JPEG for comparable image quality loss. To achieve this, it uses a wavelet transformation on the image, which takes much more computing power.
LED Light Emitting Diode.
LUT A *Look Up Table* is a block of logic in a CLB that uses SRAM technology to implement asynchronous digital logic.
MPEG Moving Pictures Expert Group. A working group of ISO/IEC in charge of the development of standards for coded representation of digital audio and video. MPEG is not an acronym for any standard; it is the acronym for the group who develops these standards that include MPEG-1 or MPEG-2 and others.
MPGA The *Metal Programmed Gate Array* family provides a low-risk conversion path from programmable gate arrays to production quantity devices. By significantly reducing the production costs of a product without technical or time-to-market risks, MPGAs prolong the life cycle of a finished design.
MSB Most Significant Bit.
PGM *Portable Graymap* format for gray scale images.
PLA A *Programmable Logic Array* is a PLD that offers flexible features for more complex designs.
PLD A *Programmable Logic Device* is an integrated circuit that can be programmed in a laboratory to perform complex functions. A PLD consists of arrays of AND and OR gates. A system designer implements a logic design with a device programmer that blows fuses on the PLD to control gate operation.
PROM A *Programmable Read-Only Memory* is a memory that can be programmed only once.
RAM Pronounced *ramm*, acronym for *Random Access Memory*, a type of computer memory that can be accessed randomly; that is, any byte of memory can be accessed without touching the preceding bytes.
RGB Image format in which image is represented in Red, Green and Blue Color planes.
RISC A *Reduced Instruction Set Computer* is a type of microprocessor that recognizes a relatively limited number of instructions.
RTL *Register Transfer Language* is a kind of Hardware Description Language (HDL) used in describing the registers of a computer or digital electronic system, and the way in which data is transferred between them.
SRAM A *Static RAM* is a device in which each bit of storage is a bistable flip-flop, commonly consisting of cross-coupled inverters. It is called “static” because it will retain a value as long as power is supplied, unlike dynamic random access memory which must be regularly refreshed.
VHDL *VHSIC Hardware Description Language* is a high-level VLSI design language. It arose out of the United State government’s Very High-Speed Integrated Circuit (VHSIC) program and is now standardized as IEEE 1076.
VHSIC *Very High-Speed Integrated Circuit* is a very high-speed computer chip which uses LSI and very large scale integration VLSI technology.
VLSI *Very Large-Scale Integration* is the process of placing thousands (or hundreds of thousands) of electronic components on a single chip.
WT Wavelet Transform.

LIST OF TABLES

Table 3-1 Virtex FPGA XCV800 features.. 17

Table 3-2 Software Tools Summary 24

Table 4-1 (2,2) CDF wavelet with lifting scheme 26

Table 4-2 Coefficients for the 4-tap Daubechies Low-pass Filter................................... 31

Table 6-1 Performance analysis of the results for implementation 45

Table 6-2 Performance analysis per pixel of the results for implementation 46

Table 6-3 Device Usage and Timing for different image sizes. 47

LIST OF FIGURES

Figure 2-1 A typical lossy image compression system.. 6

Figure 2-2 Pyramidal Decomposition of an image.. 9

Figure 2-3 Pyramidal Decomposition of 'barbara' image 10

Figure 2-4 Self-similarity of coefficients in sub-bands is exploited in EZT. 12

Figure 2-5 Identifying Zero Trees.. 13

Figure 3-1 2-slice Virtex CLB (Courtesy of Xilinx Inc.). 16

Figure 3-2 XSV800 Board Layout... 21

Figure 3-3 FPGA SRAM Block Diagram [10] 22

Figure 3-4 SRAM Write Cycle Timing Diagram [16].. 23

Figure 3-5 SRAM Read Cycle Timing Diagram [16] 23

Figure 4-1 One-dimensional Wavelet Transform 27

Figure 4-2 Three-level decomposition for Wavelet Transform....................................... 28

Figure 4-3 Wavelet Transform Implementation 29

Figure 4-4 Daubechies 4-tap Scaling and Wavelet Functions... 31

Figure 5-1 Coefficient ordering along x (row) direction 35

Figure 5-2 Coefficient ordering along y (col) direction... 36

Figure 5-3 Mallot ordering along the Level 1 of wave-letting 37

Figure 5-4 Hardware implementation scheme... 38

Figure 5-5 Top Level view of XSV board prototyping hardware 39

Figure 6-1 Timing of RAM READ control signals 42

Figure 6-2 Timing of RAM WRITE control signals 43

Figure 6-3 Timing of the Wavelet Transform process signals 43

Figure 6-4 Performance comparisons between different picture sizes. 45

Figure 6-5 Performance comparison between different picture sizes.............................. 46

ABSTRACT

Wavelet Transform has been successfully applied in different fields, ranging from pure mathematics to applied sciences. Numerous studies carried out on Wavelet Transform have proven its advantages in image processing and data compression. Recent progress has made it the basic encoding technique in data compression standards. Pure software implementations of the Discrete Wavelet Transform, however, appear to be the performance bottleneck in real-time systems. Therefore, hardware acceleration of the Discrete Wavelet Transform has become a topic of interest. The goal of this work is to investigate the feasibility of hardware acceleration of Discrete Wavelet Transform for image compression applications, and to compare the performance improvement against the software implementation. In this thesis, a design for efficient hardware acceleration of the Discrete Wavelet Transform is proposed. The hardware is designed to be integrated as an extension to custom-computing platform and can be used to accelerate multimedia applications as JPEG2000 or MPEG-4.

CHAPTER 1

INTRODUCTION

1.1 Motivation for Research

A majority of today's Internet bandwidth is estimated to be used for images and video [4]. Recent multimedia applications for handheld and portable devices place a limit on the available wireless bandwidth. The bandwidth is limited even with new connection standards. JPEG image compression that is in widespread use today took several years for it to be perfected. Wavelet based techniques such as JPEG2000 for image compression has a lot more to offer than conventional methods in terms of compression ratio. Currently wavelet implementations are still under development lifecycle and are being perfected. Flexible energy-efficient hardware implementations that can handle multimedia functions such as image processing, coding and decoding are critical, especially in hand-held portable multimedia wireless devices.

1.2 Background

Computer data compression is, of course, a powerful, enabling technology that plays a vital role in the information age. Among the various types of data commonly transferred over networks, image and video data comprises the bulk of the bit traffic. For example, current estimates indicate that image data take up over 40% of the volume on the Internet. [4] The explosive growth in demand for image and video data, coupled with delivery bottlenecks has kept compression technology at a premium. Among the several compression standards available, the JPEG image compression standard is in wide spread use today. JPEG uses the Discrete Cosine Transform (DCT) as the transform, applied to 8-by-8 blocks of image data. The newer standard JPEG2000 is based on the Wavelet Transform (WT). Wavelet Transform offers multi-resolution image analysis, which appears to be well matched to the low level characteristic of human vision. The DCT is essentially unique but WT has many possible realizations. Wavelets provide us with a basis more suitable for representing images. This is because it cans represent information at a variety of scales, with local contrast changes, as well as larger scale structures and thus is a better fit for image data.

Field programmable gate arrays (FPGAs) provide a rapid prototyping platform. FPGAs are devices that can be reprogrammed to achieve different functionalities without incurring the non-recurring engineering costs typically associated with custom IC fabrication. In this work, DWT architecture is implemented on a reconfigurable FPGA hardware. The target platform is the Xilinx Virtex FPGA. The design is based on the multi-level decomposition implementation of the Discrete Wavelet Transform. The design utilizes various techniques and specific features of the Xilinx Virtex XSV FPGA to accelerate the computation of the transform. Performance analysis includes the investigation of the performance enhancement due to the hardware acceleration. It is expected that the proposed design can substantially accelerate the DWT and the inherent scalability can be exploited to reach a higher performance in the future. The implementation can be easily modified to act as a co-processing environment for wavelet compression/decompression or even as a part of the algorithms to be used in future mobile devices for image encoding/decoding using wavelets. One drawback of the FPGA, however, is that due to the rather coarse grained reconfigurable blocks, an implementation on an FPGA is often not as efficient, in terms of space and time, as on a custom IC

1.3 Organization of Thesis

In this thesis, Chapter 2 gives a background discussion on wavelet transform based image compression. Chapter 3 introduces the FPGA programmable logic and describes the tools and methodologies involved. This chapter also describes in depth the Xilinx XSV Prototype board being used in this research. Chapter 4 presents the software implementation of DWT on images in one-dimension and multi-dimension. Chapter 5 explains the hardware implementation of the wavelet transform scheme using FPGAs. Chapter 6 summarizes the VHDL implementation and summarizes the hardware testing results are discussed. Finally Chapter 7 provides the conclusions and future extensions of work.

CHAPTER 2

WAVELET TRANSFORM BASED IMAGE COMPRESSION

Many evolving multimedia applications require transmission of high quality images over the network. One obvious way to accommodate this demand is to increase the bandwidth available to all users. Of course, this "solution" is not without technological and economical difficulties. Another way is to reduce the volume of the data that must be transmitted. There has been a tremendous amount of progress in the field of image compression during the past 15 years. In order to make further progress in image coding, many research groups have begun to use wavelet transforms.

In this chapter, we will briefly discuss image compression, the nature of wavelets, and some of the salient features of image compression technologies using wavelets. Since this is a very rapidly evolving field, only the basic elements are presented.

2.1 Introduction

Image compression is different from binary data compression. When binary data compression techniques are applied to images, the results are not optimal. In lossless compression, the data (such as executables, documents, etc.) are compressed such that when decompressed, it gives an exact replica of the original data. They need to be exactly reproduced when decompressed. For example, the popular PC utilities like Winzip or and Adobe Acrobat perform lossless compression. On the other hand, images need not be reproduced exactly. A 'good' approximation of the original image is enough for most purposes, as long as the error between the original and the compressed image is tolerable. Lossy compression techniques can be used in this application. This is because images have certain statistical properties, which can be exploited by encoders specifically designed for them. Also, some of the finer details in the image can be sacrificed for the sake of saving bandwidth or storage space.

In digital images the neighboring pixels are correlated and therefore contain redundant information. Before the image is compressed, the pixels, which are correlated is to be found. The fundamental components of compression are redundancy and irrelevancy reduction. Redundancy means duplication and irrelevancy means the parts of signal that will not be noticed by the signal receiver, which is the Human Visual System (HVS). There are three types of redundancy that can be identified:

Spatial Redundancy *is the correlation between neighboring pixel values.*

Spectral Redundancy *is the correlation between different color planes or spectral bands.*

Temporal Redundancy *is the correlation between adjacent frames in a sequence of images (in video applications).*

Image compression focuses on reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies. Since our work focuses on still image compression, therefore temporal redundancy is not discussed.

2.2 Principles of Image Compression

A typical lossy image compression scheme is shown in Figure 2-1. The system consists of three main components, namely, the source encoder, the quantizer, and the entropy encoder. The input signal (image) has a lot of redundancies that needs to be removed to achieve compression. These redundancies are not obvious in the time domain. Therefore, some kind of transform such as discrete cosine, fourier, or wavelet transform is applied to the input signal to bring the signal to the spectral domain. The spectral domain output from the transformer is quantized using some quantizing scheme. The signal then undergoes entropy encoding to generate the compressed signal. The wavelet transform mainly applies on the source encoder component portion and this is the focus of this project.

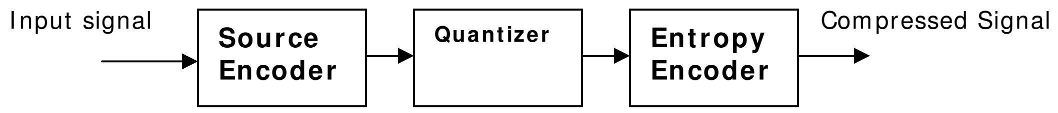


Figure 2-1 A typical lossy image compression system.

Source Encoder

An encoder is the first major component of image compression system. A variety of linear transforms are available such as Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), and Discrete Wavelet Transform (DWT). The Discrete Wavelet Transform is main focus of our work.

Quantizer

A quantizer reduces the precision of the values generated from the encoder and therefore reduces the number of bits required to save the transform co-coefficients. This process is lossy and quantization can be performed on each individual coefficient. This is known as Scalar Quantization (SQ). If it is performed on a group of coefficients together then it is called Vector Quantization (VQ).

Entropy Encoder

An entropy encoder does further compression on the quantized values. This is done to achieve even better overall compression. The various commonly used entropy encoders are the Huffman encoder, arithmetic encoder, and simple run-length encoder. For improved performance with the compression technique, it's important to have the best of all the three components.

2.3 Wavelet Transform as the Source Encoder

Just as in any other image compression schemes the wavelet method for image compression also follows the same procedure. The discrete wavelet transform constitutes the function of the source encoder. The theory behind wavelet transform is discussed below.

2.3.1 Measuring Frequency Content by Wavelet Transform

Wavelet transform is capable of providing the time and frequency information simultaneously. Hence it gives a time-frequency representation of the signal. When we are interested in knowing what spectral component exists at any given instant of time, we want to know the particular spectral component at that instant. In these cases it may be very beneficial to know the time intervals these particular spectral components occur. Wavelets (small waves) are functions defined over a finite interval and having an average value of zero. The basic idea of the wavelet transform is to represent any arbitrary function f(t) as a superposition of a set of such wavelets or basis functions. These basis functions are obtained from a single wave, by dilations or contractions (scaling) and translations (shifts). The discrete wavelet transform of a finite length signal x(n) having N components, for example, is expressed by an N x N matrix similar to the discrete cosine transform .

2.3.2 Wavelet-based Compression

Digital image is represented as a two-dimensional array of coefficients, each coefficient representing the brightness level in that point. We can differentiate between coefficients as more important ones, and lesser important ones. Most natural images have smooth color variations, with the fine details being represented as sharp edges in between the smooth variations. Technically, the smooth variations in color can be termed as low frequency variations, and the sharp variations as high frequency variations.

The low frequency components (smooth variations) constitute the base of an image, and the high frequency components (the edges which give the details) add upon them to refine the image, thereby giving a detailed image. Hence, the smooth variations are more important than the details.

Separating the smooth variations and details of the image can be performed in many ways. One way is the decomposition of the image using the discrete wavelet transform. Digital image compression is based on the ideas of sub-band decomposition or discrete wavelet transforms. Wavelets which refer to a set of basis functions are defined recursively from a set of scaling coefficients and scaling functions. The DWT is defined using these scaling functions and can be used to analyze digital images with superior performance than classical short-time Fourier-based techniques, such as the DCT. The basic difference between wavelet-based and Fourier-based techniques is that short-time Fourier-based techniques use a fixed analysis window, while wavelet-based techniques can be considered using a short window at high spatial frequency data and a long window at low spatial frequency data. This makes DWT more accurate in analyzing image signals at different spatial frequency, and thus can represent more precisely both smooth and dynamic regions in image. The compression system includes forward wavelet transform, a quantizer, and a lossless entropy encoder. The corresponding decompressed image is formed by the lossless entropy decoder, a de-quantizer, and an inverse wavelet transform. Wavelet-based image compression has good compression results in both rate and distortion sense.

2.4 Wavelet Decomposition

There are several ways wavelet transforms can decompose a signal into various sub bands. These include uniform decomposition, octave-band decomposition, and adaptive or wavelet-packet decomposition. Out of these, octave-band decomposition is the most widely used.

The procedure is as follows: wavelet has two functions “wavelet “and “scaling function”. They are such that there are half the frequencies between them. They act like a low pass filter and a high pass filter. Figure 2-2 shows a typical decomposition scheme. The decomposition of the signal into different frequency bands is simply obtained by successive high pass and low pass filtering of the time domain signal. This filter pair is called the analysis filter pair. First, the low pass filter is applied for each row of data, thereby getting the low frequency components of the row. But since the low pass filter is a half band filter, the output data contains frequencies only in the first half of the original frequency range. By Shannon's Sampling Theorem, they can be sub-sampled by two, so that the output data now contains only half the original number of samples. Now, the high pass filter is applied for the same row of data, and similarly the high pass components are separated.

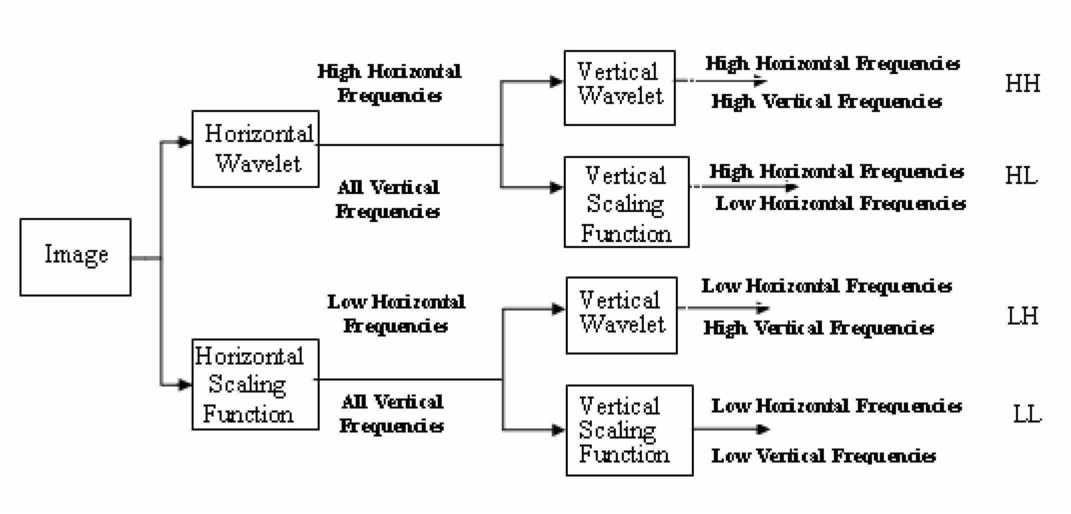
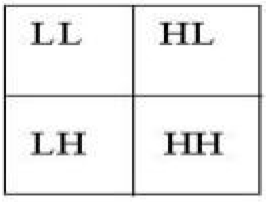
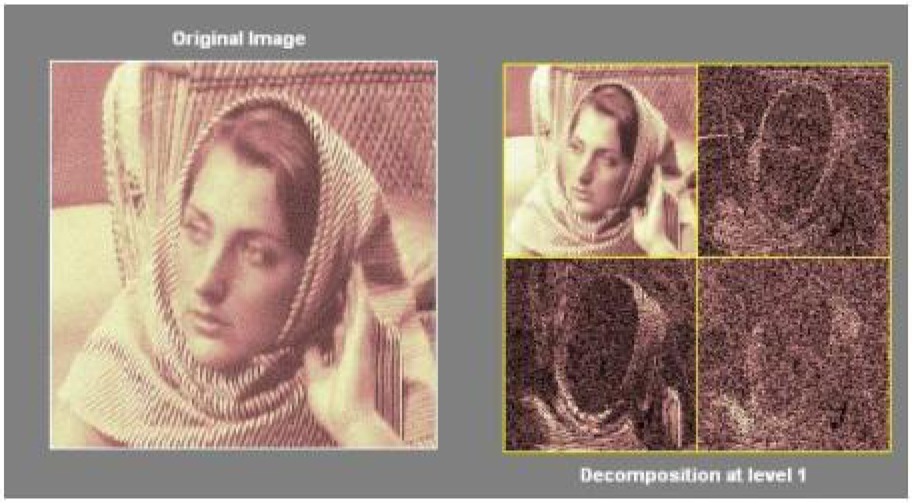


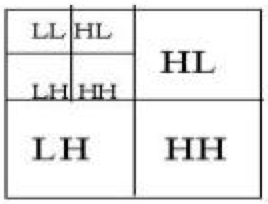
Image
Horizontal Wavelet
Horizontal Scaling Function
High Horizontal Frequencies
All Vertical Frequencies
Low Horizontal Frequencies
All Vertical Frequencies
Vertical Wavelet
Vertical Scaling Function
Vertical Wavelet
Vertical Scaling Function
High Horizontal Frequencies
High Vertical Frequencies
HH
High Horizontal Frequencies
Low Horizontal Frequencies
HL
Low Horizontal Frequencies
High Vertical Frequencies
LH
Low Horizontal Frequencies
Low Vertical Frequencies
LL

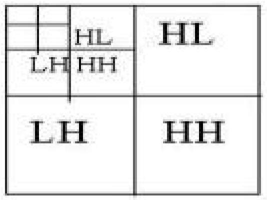
Figure 2-2 Pyramidal Decomposition of an image

This is a non-uniform band splitting method that decomposes the lower frequency part into narrower bands and the high-pass output at each level is left without any further decomposition. This procedure is done for all rows. Next, the filtering is done for each column of the intermediate data. The resulting two-dimensional array of coefficients contains four bands of data, each labeled as LL (low-low), HL (high-low), LH (low-high) and HH (high-high). The LL band can be decomposed once again in the same manner, thereby producing even more sub bands. This can be done up to any level, thereby resulting in a pyramidal decomposition as shown in figure 2-2.









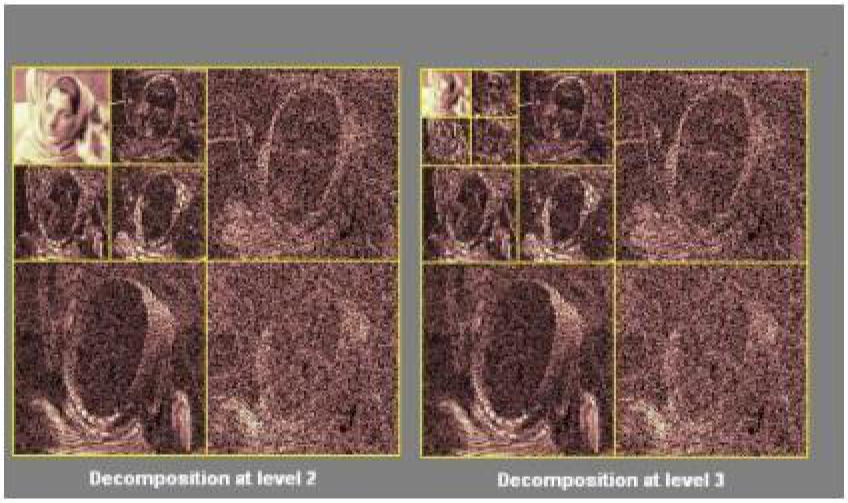


Figure 2-3 Pyramidal Decomposition of 'barbara' image

The LL band is decomposed thrice in figure 2-2. The compression ratios with wavelet-based compression can be up to 300-to-1, depending on the number of iterations. The LL band at the highest level is most important, and the other 'detail' bands are of lesser importance, with the degree of importance decreasing from the top of the pyramid to the bands at the bottom. This can be done to any image. Figure 2-3 shows, how it would work for an image.

2.5 Embedded Zero Tree Encoding Technique

As discussed in the principles of image compression, the image compression scheme using wavelet transform consists of the following procedure:

- Decomposition of signal using filters banks.
- Down sampling of Output of filter bank
- Quantization of the above
- Finally encoding

The discrete wavelet transform constitutes the source encoder block. Wavelet transform based image compression is different from other techniques in the sense that sometimes the quantization and the encoding steps are merged in techniques such as the Embedded Zero Tree encoding. This technique is briefly discussed here.

The Embedded Zero tree (EZT) wavelet algorithm was invented by Shapiro in 1993. Basically a wavelet transform on an image leads to the formation of sub bands. The EZT algorithm exploits the self similarity of coefficients in different wavelet bands. The algorithm is in such a way that it takes care of the intermediate quantization process. In this scheme, the lower rate codes or the averages (LL) are embedded at the beginning of the bit stream. The functioning of this algorithm is briefly as follows. EZW is based on the hypothesis that "if a wavelet coefficient at a coarse scale (parent) is insignificant with respect to a given threshold T, then all wavelet coefficients of the same orientation in the same spatial location at finer scale (children) are likely to be insignificant with respect to T." This idea is shown in figure 2-4.

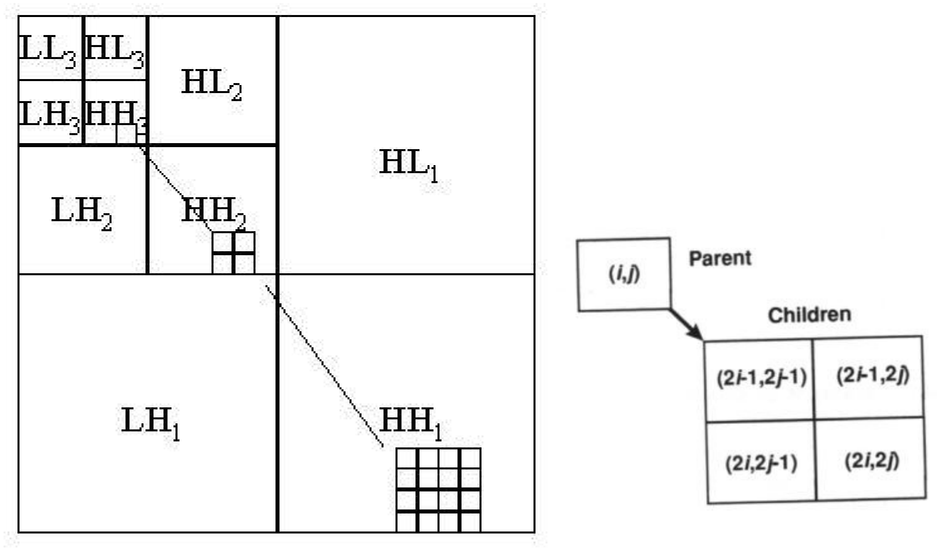


Figure 2-4 Self-similarity of coefficients in different wavelet bands (sub-bands) is exploited in EZT.

The parent for the example in figure 2-4 is in the level HH3 whose children is HH2. Level HH2 in turn is the parent to the sub band level HH1. Although the wavelet transform decorrelates image information within a subband, there is still a high degree of correlation between subbands. EZW encodes zeros, rather than the data. A zero at a coarse scale or a parent level is a good predictor of zeros at a finer scale. This method can encode a lot of information in specifying just the root of a zero-tree. The chances of finding a zero tree increases with a coarser quantization. At a coarser quantization more lower level values are set to zero. In successive quantization, quantize the (remaining) image data using successively finer quantization steps. This leaves a binary image to encode at each step whose values are either 0 or 1. This encoding using zero trees completely specifies a binary image in which successive zero-trees encode less significant bits. Thus to find and scan through children pixel at each level, this algorithm needs random access to memory. The algorithm is complex as the generation of binary trees needs extensive recursion. Figure 2-5 shows the process of identifying zero trees.

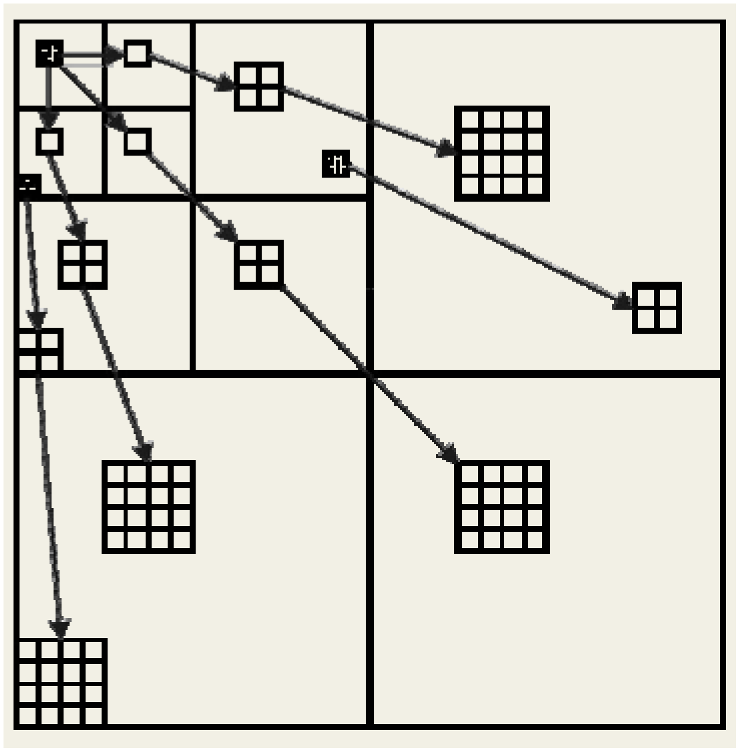


Figure 2-5 Identifying Zero Trees.

The successive quantization algorithm for embedded zero tree encoding is as follows:

- *1. Choose a high quantisation step T (half the max value)*
- *2. Until finest quantisation step, repeat*
    - *2.1 Quantise wavelet coefficients using T*
        - *ie. if* $(w(x,y)<T)$ $w'(x,y)=0$ *; else* $w'(x,y)=1$*;*
        - *Creates a binary image of the most significant wavelet coefficients*
    - *2.2 Encode binary image (w') using zero-trees and output*
        - *(This is the highest order bits of the remaining data)*
    - *2.3 Subtract the encoded data from the remaining data*
        - $w(x,y)=w(x,y) - w'(x,y)*T$
    - *2.4 half the quantisation step size*
        - $T=T/2$
- *3. End*

CHAPTER 3

FPGA TOOLS AND METHODS

As the first aim of this work is testing the feasibility and performance of an algorithm, great flexibility and versatility are required in the design description. The tools and methodology being used are able to help designers deal with architectural complexity, and are capable of system parameterization and result comparisons. Moreover, the tools should limit the time and resources needed for circuit verification. There are two basic hardware design methodologies currently available: schematic and language-based designs. The schematic based design, despite of better optimization implementation in terms of both area and speed, does not fulfill the requirements of simplicity and speed of designing stated above. Nevertheless, the language based counterpart suffers from the synthesis tool used as well as the code style with which designs being synthesized are written, as is the case for software developing compilers. These two factors can potentially lead to variances when comparing synthesis tool outputs. For the purpose of this work the language based design was chosen. The language based design choice relies primarily on two components: the hardware side, represented by the FPGA technology, and the programming language counterpart, represented for the language chosen, i.e. VHSIC Hardware Description Language (VHDL). The former is discussed in the first section, where the more relevant features, from this project point of view, will be illustrated; the latter is introduced in the next section which looks at basic and fundamental concepts in VHDL.

3.1 FPGA Programmable Logic

The field-programmable gate array (FPGA) is a type of programmable device. Programmable devices are a class of general-purpose chips that can be configured for a wide variety of applications, having capability of implementing the logic of hundreds or thousands of discrete devices. Programmable read-only memory (PROM), erasable programmable read-only memory (EPROM) and electrically erasable PROM (EEPROM) are the oldest members of that class, while programmable logic device (PLD) and programmable logic array (PLA) represent more recent attempts to provide the end-user with an on-site customization using programming hardware. This technology, however, is limited by the power consumption and time delay typical of these devices. In order to address these problems and achieve a greater gate density, metal programmed gate arrays (MPGA) were introduced to programmable logic industry market. An MPGA consists of a base of pre-designed transistors with customized wiring for each design. The wiring is built during the manufacturing process, so each design requires expensive custom masks and long turnaround time. Since FPGAs are application-specific ICs it is often argued that FPGA is an application specific integrated circuit (ASIC) technology.

3.1.1 FPGA Benefits

FPGAs offer the benefits of both PLD and MPGA. In fact, the computing core of an FPGA consists, in general, of a highly complex re-programmable matrix of logic IC, registers, RAM and routing resources. These resources can be used for performing logical and arithmetical operations, for variable storage and to transfer data between different parts of the system. Furthermore, since there is no centralized control and sequential instructions to process, typically thousands of operations can be performed in parallel on an FPGA during every clock cycle. Though the clock speed of FPGAs (20- 130MHz) is lower than of current RISC systems (100-500MHz), the resulting performances are comparable in many applications such as image and video processing and data encryption.

3.1.2 SRAM-based FPGA

The SRAM FPGA gets its name from the fact that programmable connections are made using pass-transistors, transmission gates, or multiplexers that are controlled by static random access memory (SRAM) cells.

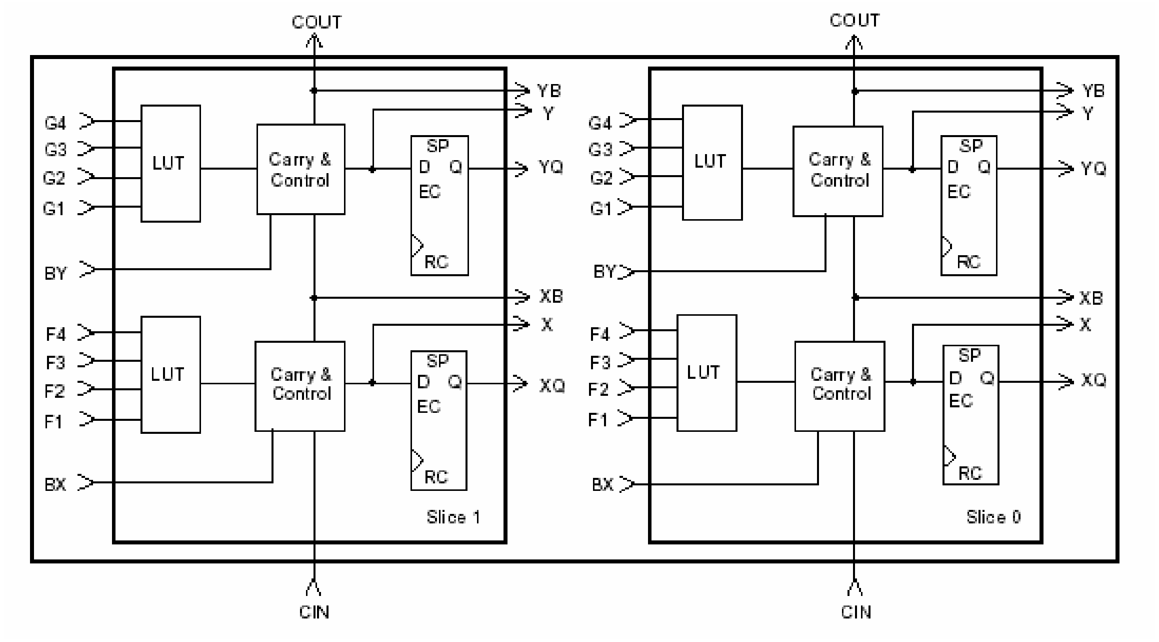


Figure 3-1. 2-slice Virtex CLB (Courtesy of Xilinx Inc.).

The advantage of this technology resides in the fact that it allows fast in-circuit reconfiguration. The major disadvantage, however, is the size of the chip required by the SRAM technology.

The FPGA has three major configurable elements: configurable logic blocks (CLBs), input/output blocks (IOBs), and interconnects. The CLBs provide the functional elements for constructing user's logic. As shown in figure 3-1, it normally comprises of a Look Up Table (LUT) with 4 inputs and a flip-flop output. The IOBs provide the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks. Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA. The XCV800, belonging to the Virtex family from Xilinx, is the FPGA chosen for this implementation. Its features, summarized in Table 3-1, are generous in gates and I/O pins allowing the designer long synthesis waits and troublesome pin assignments.

Table 3-1 Virtex FPGA XCV800 features

System gates	888,439
CLB Arrays	56 x 84
Logic Cells	21,168
Maximum I/Os	512
BlockRAM Bits	114,688
Flip-Flop Count	43,872

Besides the common CLB structure depicted above, this FPGA family provides digital delay-locked loops (DLLs), dedicated electronics to deal with clock distribution problems, and tristate buffers (BUFTs), for driving on-chip busses. A major feature is the Virtex Block SelectRAM+ technology. A Block SelectRAM+ is a real synchronous four CLBs high memory block which extends the full height of the chip, immediately adjacent to a CLB column location. Each cell can be seen as a fully synchronous dual-ported 4096 bit RAM with independent control signals for each port. This structure is suitable to make it work like an internal memory bank. The hardware platform used, XESS XSV Virtex Board, is a stand alone board with one Xilinx XCV800 FPGA.

3.2 HDL-VHDL

VHSIC Hardware Description Language (VHDL) is a language for describing digital electronic systems. Its nested acronym is due to the fact that it arose out of the United State government's Very High-Speed Integrated Circuit (VHSIC) program and to the need for a standard and well-defined language to describe digital integrated circuits (ICs). It was subsequently embraced by the IEEE, which adopted it in the form of the IEEE Standard 1076 Standard VHDL Language Reference Manual in 1987 and revised it in 1992. This is the latest official version and is often referred to as VHDL-93. The main difference between VHDL and another programming language more widely used in computer science as, for example, 'C', is that VHDL enables the execution of concurrent statements. In contrast with a sequential statement, a concurrent statement is so called because conceptually it can be activated to perform its task together with any other concurrent statements. Concurrency is useful for modeling the way real circuits behave (you can think of any logic port such an SR flip-flop), but can potentially lead a neophyte to misunderstandings and unpredicted results. The real power of VHDL, resides in the possibility to look at the same system with different levels of abstraction, where different pieces of information play different roles in describing that system's model. If we start with a requirement document for our system, we can decompose it in several components that jointly fulfill the requirement proposed. Each of these components can be in turn decomposed into subcomponents, for which design we need to pay attention to a smaller amount of relevant information than being overwhelmed by masses of detail. Moreover, the design we are focusing on can be described using abstract data types or going closer to the electronic rules. The result of this process is a hierarchically composed system, built from primitive elements. VHDL attains this goal with a rigorous formalism that comprises three fundamental concepts: entity, architecture and configuration.

3.2.1 Entities

In VHDL, an entity is the equivalent of an IC package in electronics. A package is defined by its name and the number and nature of ports it uses to exchange data and interact with external circuits. It is not relevant to the function that the component performs, the resources it uses or the complexity by which it is characterized. What matters is, using the language's terminology, the number of ports, by which information is fed into and out the circuit, the type, which specifies the kind of information that can be communicated, and the mode which specifies whether the information flows into or out through the entity's ports. At this point it is important to have clear in mind the function and number of signals for the entity being defined. Otherwise, when all the components are assembled together to form the whole system, the code can be subjected to several and problematic changes. Stated in another way, the system has to be carefully split in subcomponents for which the major number of details has to be planned to avoid subsequently cumbersome fixes. The entity concept easily allows the programmer to apply the divide and conquer rule not only because a complex system can be split in smaller and more tractable sub-systems, but since it also permits to face structural problems explained above during a well-distinct phase and postpone descriptive problems to a latter stage by using architectures.

3.2.2 Architectures

Architecture, continuing with the analogy of an IC, is equivalent to the internal electronic circuit that performs the function for which the component has been designed. At this stage all the signals the entity uses to communicate with the external world are defined and unchangeable. Inside the entity, instead, the programmer can write any type of statements allowed by the VHDL syntax, like other components, representing sub-circuits, other signals, to connect them, or processes, which contain sequential statements operating on values. The purpose is only that to apply some operations to data on input ports and generate some results to assign to output ports. Within architecture the programmer can concentrate on the descriptive part of the system, but the divide and conquer motto can be applied once again. VHDL, indeed, offers the opportunity to choose the desired abstraction level. Thus, for a first behavioral version, aiming to investigate a circuit feasibility or correctness, code lines containing high level data types or cumbersome mathematical formulas and few electronic details will be advisable. Then, a subsequent description can go further and optimize the implementation with clever solutions for better performance, smaller area and fulfillment of constraints.

3.2.3 Configurations

In the previous two subsections we saw that the same entity can be usefully described, unless the functional behaviors match, with different approaches that correspond to different architectures. Due to this correspondence, nothing prevents us from choosing architecture of an entity regardless of architectures chosen for other entities. The configuration declaration permits of assembling the system based on our goal, architectures availability, or simply taste. To clarify this idea, let us assume we have n entities {E1,…, E_n}. Let also suppose each of them has got two architectures, $A^i{}_1$ and $A^i{}_2$ with i in {1,…,n}. If the entire system is only composed of all the entities

$\{E1,\ldots,E_n\}$, combining the two architectures for each of them we can virtually have 2^n different representation of the same system.

In practice, the situation can be slightly different. Indeed, we can have a behavioral and abstract architecture that does not use the clock signal, common to almost all the digital systems. Such an asynchronous architecture cannot be sided by a synchronous version of another component, because they do not share a piece of information carried by the clock signal. Nevertheless, the configuration facility is highly attractive since a system that comprises of only behavioral architectures can act as reference model for correctness in successive implementations.

3.2.4 Generic Parameters

The *generic interface* list is another powerful tool provided by VHDL language. Its aim is to allow parameterized descriptions of entities. In fact, from an architecture point of view, a generic element is seen as a constant whose visibility extends into the whole architecture body corresponding to that entity declaration. The actual generic value, on the other hand, is defined in the entity that lies on a upper level in the hierarchy and is passed down to the component as a parameter. To demonstrate the concept, we can consider a situation where we need a 2- input multiplexer for which we design architecture. If subsequently we need a multiplexer with 4 inputs and do not use the generic facility, we have to write a new entity with two more pins and a new architecture. Instead, with a generic element in the entity definition and an opportune descriptive code, we can change only one parameter during the component instantiation and still use the same multiplexer implementation. This approach has several advantages. The entity can result in a more dynamic and versatile component. The reduced number of lines permits handling with less architectures. It also simplifies code maintenance since, if a logical error is found, only one implementation needs being modified. At last but not least, a parametric description normally has a more regular structure that may be better synthesized by the synthesis tool. The drawback is the increased complexity in the architecture design which requires better summarizing skills and wider perspectives. In fact, a parametric entity leads the programmer to face problems due to out-of-range values or exceptions not contemplated in the original version. In the end, especially after some practice, the advantages will certainly prevail.

3.3 XESS XSV Virtex FPGA Board

The XESS XSV board [10] contains a CPLD and a XCV800 FPGA with 800,000 gates. It has the capability of processing audio and video signals. It can output these video signals through its 110MHz, 24-bit RAMDAC. It also contains two independent RAM banks of 512K x 16 SRAM used for local buffering of data and signals. The XSV800 Board has a variety of interfaces: serial and parallel ports, Xchecker cable, USB port, PS/2 keyboard and mouse ports, and 10/100 Ethernet PHY layer interface. There are also two independent expansion ports. Figure 3-2 shows the layout of the XSV800 board.

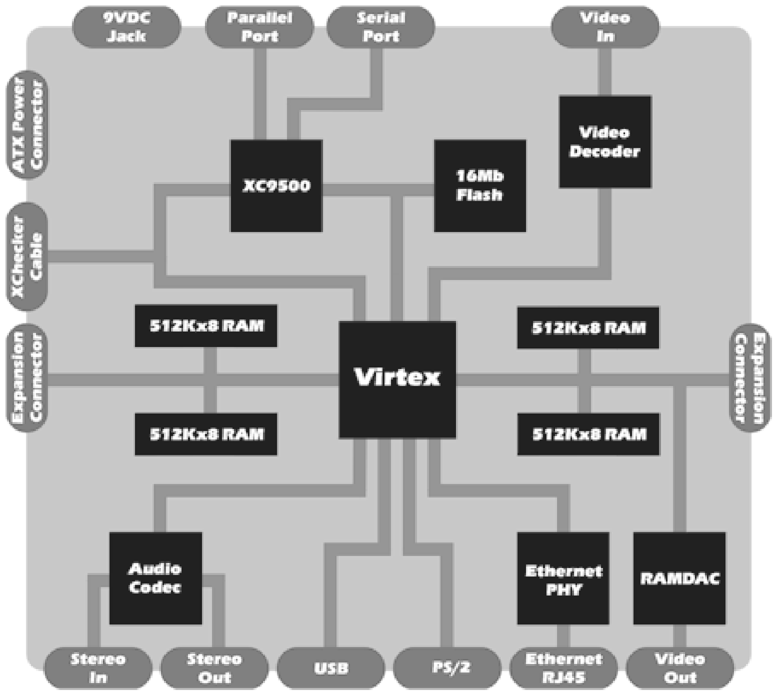


Figure 3-2 XSV800 Board Layout

FPGA: The Virtex FPGA is a programmable logic device that transmits/receives address and data signals to and from the two independent SRAM banks. It also conveys control information to the both SRAM banks using the clock. The FPGA can also generate a video signal that can be displayed on a VGA monitor using a BT481A RAMDAC. [10]

SRAM: The FPGA has access to two independent banks of SRAM. Each SRAM bank is organized as 512K × 16 bits. Each bank is made from two Alliance AS7C4096 memory chips of organization 524,288 words x 8 bits. Each of the 4 SRAM chips is directly connected to the FPGA as shown in figure 3-3. Two each of them form the left memory bank and the right memory banks, giving 1MB RAM each for the left bank and the right bank. The embedded memory is accessible for read/write from both the computer (through the XSV SRAM Utility) as well as from the FPGA. Each of the 1MB memory bank is organized as 524288 words of 16 bits each. The image data is stored in the left SRAM bank. The FPGA reads and process the image and writes the result to the right SRAM bank.

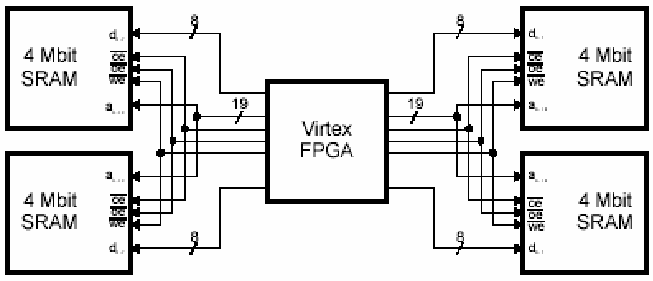


Figure 3-3 FPGA SRAM Block Diagram [10]

A write cycle is accomplished by asserting write enable (WE) and chip enable (CE). Data on the input pins I/O1–I/O8 is written on the rising edge of WE (write cycle 1) or CE (write cycle 2). To avoid bus contention, external devices should drive I/O pins only after outputs have been disabled with output enable (OE) or write enable (WE). Figure 3-4 below illustrates the idea.

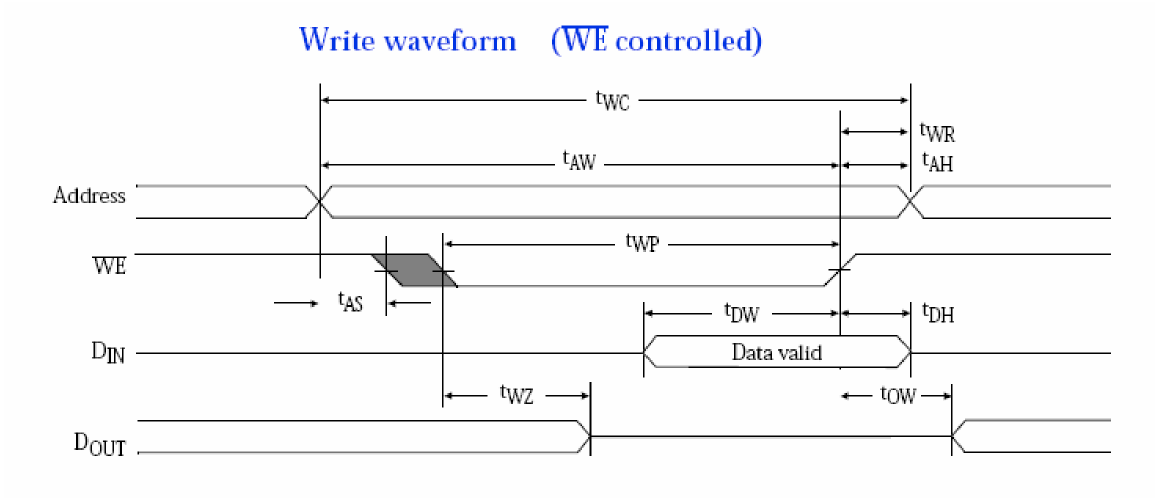


Figure 3-4 SRAM Write Cycle Timing Diagram [16]

A read cycle is accomplished by asserting output enable (OE) and chip enable (CE), with write enable (WE) high. The chip drives I/O pins with the data word referenced by the input address. When either chip enable or output enable is inactive, or write enable is active, output drivers stay in high-impedance mode. Figure 3-5 below illustrates the idea. [16]

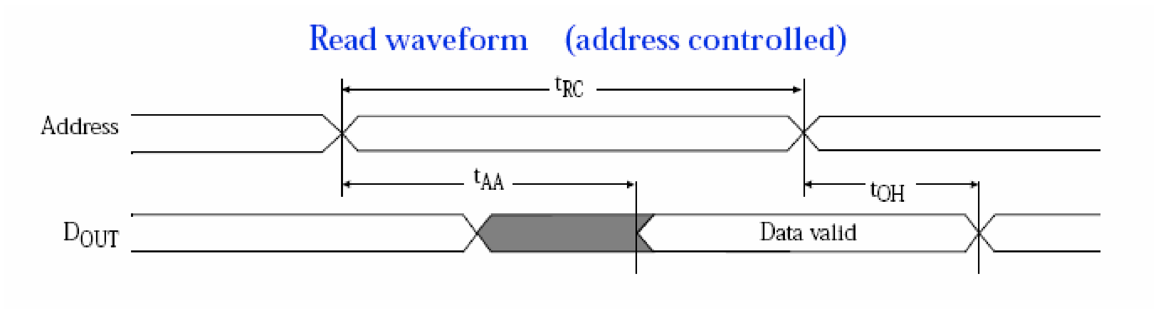


Figure 3-5 SRAM Read Cycle Timing Diagram [16]

3.4 Software Tools

The following section briefly describes the software tools used to write, compile, test and synthesize the project. To develop, test, and debug the project several commercial programs played an important role**.** Table 3.2 provides a summary of the software tools used in this work.

Table 3-2 Software Tools Summary

Software Implementation	**MATLAB Version 6.5**
Development Environment	**Xilinx ISE Version 6.1 supported editing compilation and simulation.**
Simulations	**Xilinx Waveform Complier and MODELSIM Version 5.7c for Xilinx ISE**
Synthesis	**Xilinx Synthesis Tool (XST) Version 6.1 and Synopsys Tool**
Place-and-Route and Back-Annotation	**Xilinx ISE Version 6.1**

Other tools used include:

XSV SRAM Utility**:** A utility developed by Xess Inc [10] to read from and write to the left and the right SRAM memory banks of the XSV board. This utility takes as an input the HEX file of data that needs to be written into the SRAM. This utility then modifies the file to add checksum and writes into either the left or the right bank of the board through the PC parallel port. This utility also can perform a read of either of the SRAM banks to read the results of the execution. This utility uses two separate configuration bit files to perform this operation.

GSXLOAD Utility: A utility developed by Xess Inc [10] to write the configuration bit stream through the parallel port into the XSV board. This also has facility to write into and read from the two SRAM data banks.

HEX WORKSHOP: This utility is used to view, edit and modify data or program files in the hex or binary mode. In this project this utility is used to view and edit the hex or bit files to inspect its contents for viewing, debugging etc.

CHAPTER 4
SOFTWARE IMPLEMENTATION

Matlab, a popular mathematical modeling software, is used for the software implementation of the discrete wavelet transform. Matlab is also for the development of software modules for converting Portable Grey Map (PGM) image files to memory files, and vice versa. The Matlab programming environment provided all the necessary functions and tools needed to achieve this. The following section details the software implementation and the different software modules.

4.1 1-D Wavelet Transform(WT)

The main intent of wavelet transform (WT) is to decompose a signal f, in terms of its basis vectors $f = \sum a_i W_i$. To have an efficient representation of signal f using only a few coefficients a_i, the basis functions should match the features of the signal we want to represent. The *(2, 2) Cohen-Daubechies-Feauveau(CDF) Wavelet* [7] is widely used for image compression because of its good compression characteristics. The original filters have 5 + 3 = 8 filter coefficients, whereas an implementation with the lifting scheme has only 2 + 2 = 4 filter coefficients. The forward and reverse filters are shown in Table 4-1. Fractional numbers are converted to integers at each stage. Though such an operation adds non-linearity to the transform, the transform is fully invertible as long as the rounding is deterministic. In Table 4.1 *x* represents the image pixel values and *s* stands for the summing or the low pass coefficients and *d* stands for the difference or the high pass coefficients.

Table 4-1 (2,2) CDF wavelet with lifting scheme.

Forward transform

s_i	$\leftarrow x_{2i}$
d_i	$\leftarrow x_{2i+1}$
d_i	$\leftarrow d_i - (s_i + s_{i+1}) / 2$
s_i	$\leftarrow s_i + (d_{i-1} + d_i) / 4$

Inverse transform

s_i	$\leftarrow s_i - d_i / 2$
d_i	$\leftarrow d_i + s_i$
x_{2i}	$\leftarrow s_i$
x_{2i+1}	$\leftarrow d_i$

Before an actual hardware implementation it is decided to implement the WT module in software. The discrete wavelet transform (DWT) has been implemented using Matlab. The wavelet transform routine process is a modified version of the biorthogonal Cohen-Daubechies–Feauveau wavelet [9]. The one-dimensional discrete wavelet transform (DWT) can be described in terms of a filter band as shown in Figure 4-1. An input signal $x[n]$ is applied to the low pass filter $l[n]$ and to the analysis high-pass filter $h[n]$. The odd samples of the outputs of these filters are then discarded, corresponding to a decimation factor of two. The decimated outputs of these filters constitute the reference signal $r[k]$ and the detail signal $d[k]$ for a new-level of decomposition. During reconstruction, interpolation by a factor of two is performed, followed by filtering using the low-pass and high-pass synthesis filters $l[n]$ and $h[n]$. Finally, the outputs of the two synthesis filters are added together.

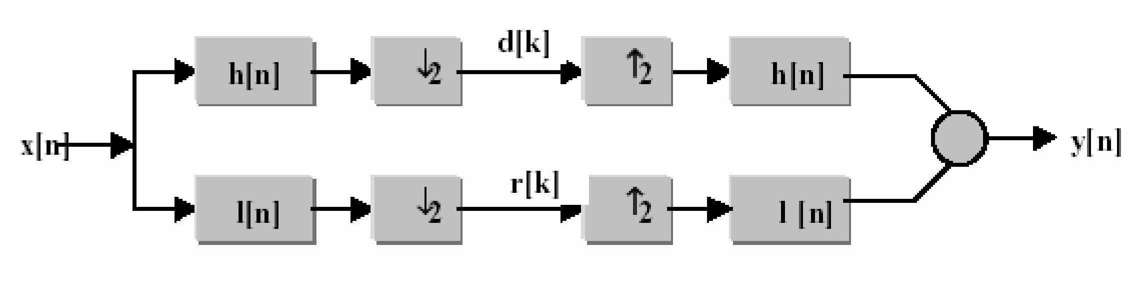


Figure 4-1 One-dimensional Discrete Wavelet Transform

The above set of filter operations can be mathematically expressed as the following equations:

$$d[k] = \sum_n x[n] \bullet h[2k - n] \tag{4.1}$$

$$r[k] = \sum_n x[n] \bullet l[2k - n] \tag{4.2}$$

$$x[k] = \sum_n (d[k] \bullet g[-n + 2k]) + (r[k] \bullet h[-n + 2k]) \tag{4.3}$$

4.2 Implementation in Multi Dimension

For a multilevel decomposition, the above process is repeated. The previous level's lower resolution reference signal $r_i[n]$ becomes the next level sub-sampling input, and its associated detail signal $d_i[n]$ is obtained after each level filtering. Figure 4-2 illustrates this procedure. The original signal $x[n]$ is input into the low-pass filter $l[n]$ and the high-pass filter $h[n]$. After three levels of decomposition, a reference signal $r_3[n]$ with the resolution reduced by a factor of 2^3 and detail signals $d_3[n]$, $d_2[n]$, $d_1[n]$ are obtained. These signals can be used for signal reconstruction.

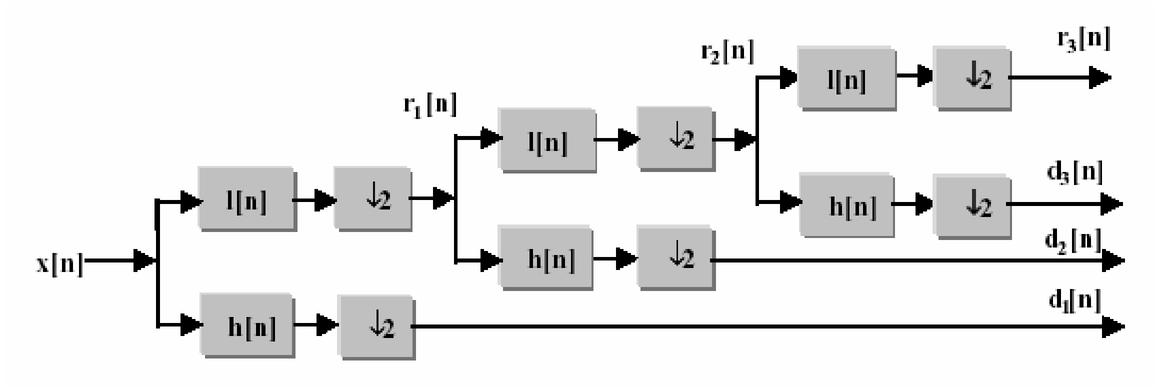


Figure 4-2 Three-level decomposition for Wavelet Transform.

The wavelet transform routine employs a shifting scheme to simplify the wavelet implementation. Therefore, it only requires integer adds and shifts, which make it easier to implement on hardware. The computation of the wavelet filter is performed according to the following equations:

$$D_0 = D_0 + D_0 - S_0 - S_0 \quad \textbf{(4.4)}$$

$$S_0 = S_0 + (2 * D_0 / 8) \quad \textbf{(4.5)}$$

$$D_i = D_i + D_i - S_i - S_{i+1} \quad \textbf{(4.6)}$$

$$S_i = S_i + ((D_{i-1} + D_i) / 8) \quad \textbf{(4.7)}$$

In the above equations, D_i and S_i are odd and even pixel values taken from one row or column, respectively. In image compression, one row or column of an image is regarded as a signal. Figure 4-3 shows the row and column formation at level 1, level 2 and level 3 for a 512 X 512 pixel image.

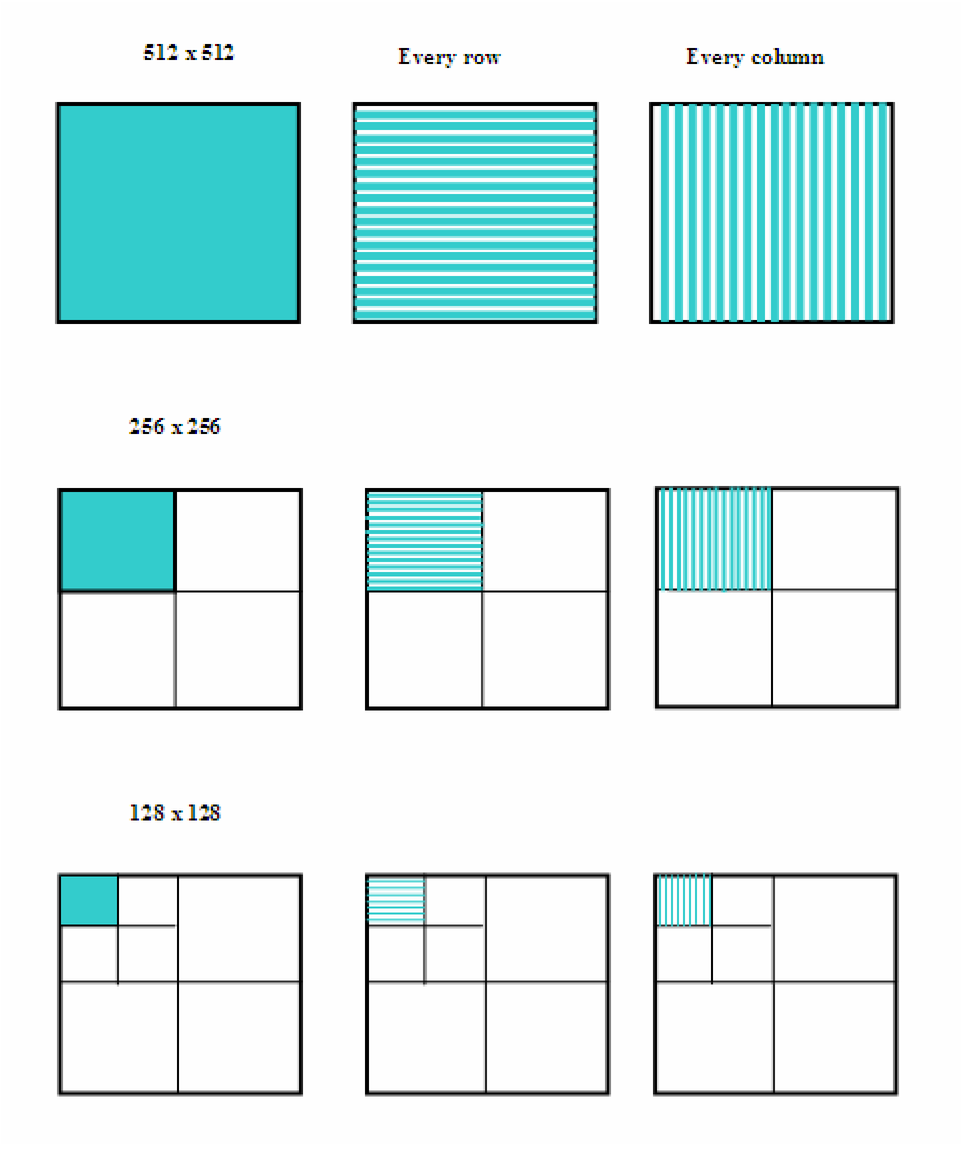


Figure 4-3 Wavelet Transform Implementation at Level 1, 2 and 3.

Every row or column is arranged and assigned in the following manner:

S0 D0 S1 D1 S2 D2 S3 D3…

For every row or column:

Repeat until end of row or column:

$$D_0 = D_0 + D_0 - S_0 - S_0$$

$$S_0 = S_0 + (2 * D_0 / 8)$$

$$D_i = D_i + D_i - S_i - S_{i+1}$$

$$S_i = S_i + ((D_{i-1} + D_i)/8)$$

The calculation of the wavelet transform requires pixels taken from one row or column at a time. In Equations (4.4) – (4.7), D_i should be calculated before processing S_i. Therefore, the odd pixel should be processed first, then the even pixel due to the data dependency. There are a total of three levels based on the 3-level decomposition wavelet transform algorithm discussed above. In each level, the rows are processed first then the columns. Each level's signal length (amount of each row/column pixels) is half of the previous level. Equations (4.4) – (4.7) are grouped into a function in the MATLAB code. This function is executed repeatedly for every row operations. After the entire set of rows is operated upon, this function is then performed on the columns. This process continues for three level hierarchies where the row/column size is decreased by a factor of 2 at each level.

4.3 Daubechies Wavelet Implementation

There are no explicit expressions for Daubechies compact support orthogonal wavelets and corresponding scaling functions. Table 4-2 presents the low-pass wavelet filter coefficients for Daubechies 4 taps wavelet [4]. Figure 4-3 shows the corresponding scaling and wavelet functions. Note that the longer the filter, the smoother the scaling and wavelet functions.

Table 4-2 Coefficients for the 4-tap Daubechies Low-pass Filter

H0	**.4829629131445341**
H1	**.8365163037378077**
H2	**.2241438680420134**
H3	**-.1294095225512603**

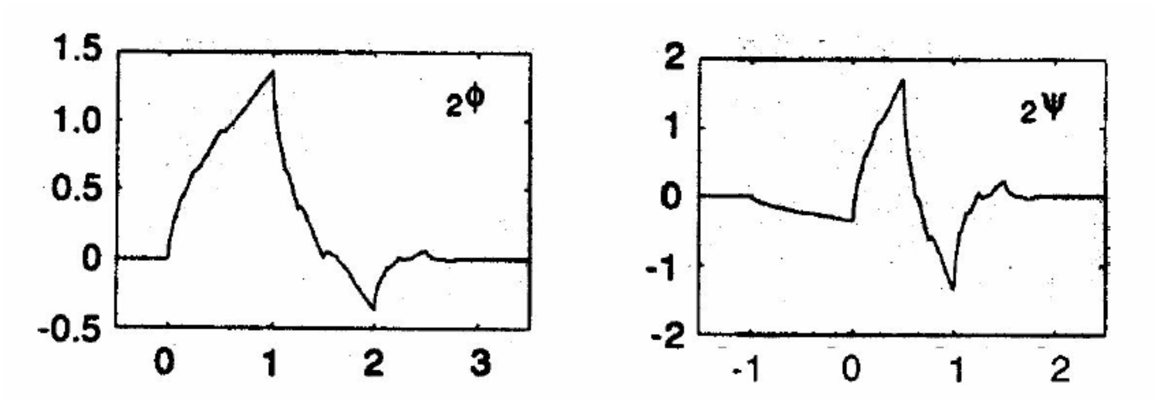


Figure 4-4 Daubechies 4-tap Scaling and Wavelet Functions.

The transform (decomposition) matrix W is given by:

$$W = \begin{bmatrix}
h0 & h1 & h2 & h3 & & & & & & & & & & & & \\
h3 & -h2 & h1 & -h0 & & & & & & & & & & & & \\
& & h0 & h1 & h2 & h3 & & & & & & & & & & \\
& & h3 & -h2 & h1 & -h0 & & & & & & & & & & \\
& & & & h0 & h1 & h2 & h3 & & & & & & & & \\
& & & & h3 & -h2 & h1 & -h0 & & & & & & & & \\
& & & & & & h0 & h1 & h2 & h3 & & & & & & \\
& & & & & & h3 & -h2 & h1 & -h0 & & & & & & \\
& & & & & & & & h0 & h1 & h2 & h3 & & & & \\
& & & & & & & & h3 & -h2 & h1 & -h0 & & & & \\
& & & & & & & & & & h0 & h1 & h2 & h3 & & \\
& & & & & & & & & & h3 & -h2 & h1 & -h0 & & \\
& & & & & & & & & & & & h0 & h1 & h2 & h3 \\
& & & & & & & & & & & & h3 & -h2 & h1 & -h0 \\
h2 & h3 & & & & & & & & & & & & & h0 & h1 \\
h1 & -h0 & & & & & & & & & & & & & h3 & -h2
\end{bmatrix}$$

where *h*[n] is the low pass filter, *g*[n] is the high pass filter, and *N* is the filter length. The discrete wavelet transform usually is implemented by using a hierarchical filter structure. One important property of the Daubechies wavelets is the coefficient relationship between the highpass and lowpass filters. They are not independent of each other, and they are related by:

$$g[N-1-n] = (-1)^n \bullet h[n]$$

This gives $g0 = h3, g1 = -h2, g2 = h1$ and $g3 = -h0$. Since Daubechies wavelets are orthogonal, the inverse matrix of *W* is the same [4].

4.4 Software Implementation Results

MATLAB programs are coded to find the DWT using both the Biorthogonal Cohen-Daubechies–Feauveau wavelet and the Daubechies Wavelet for test images. The image format used is the Portable Grey Map (PGM) format in which pixels are stored in the unsigned char type, providing a maximum of 256 gray scale levels or 8-bit data per pixel. The grayscale format was chosen to keep the algorithm fairly simple compared to processing for RGB 3 color or more planes. Also no special boundary treatment is performed on the images while evaluating the DWT.

4.4.1 Biorthogonal Cohen-Daubechies–Feuvear wavelet

The test image used is the 512 X 512 pixel image *'barbara.pgm'*. The Biorthogonal Cohen-Daubechies–Feuvear wavelet acts as a series of additions and subtractions to generate the average and detail coefficients. This is seen in the program *'wavelet_t1.m'* in Appendix A. Figure A1 shows the original image '*barbara.pgm'*. Figure A2 shows the wavelet transform on the image at the level 1 decomposition. Figure A3 shows the wavelet transform on the image at the level 3 decomposition.

4.4.2 Daubechies wavelet

The test image used is the 512 X 512 pixel image *'boat.pgm'*. The Daubechies wavelet acts as a series of high and low pass filter to generate the average and detail coefficients. This is seen in the program *'wavelet_t1.m'*. Figure A4 (of Appendix A) shows the original image '*boat.pgm'*. Figure A5 shows the Daubechies wavelet transform on the image at the level 1 decomposition. Figure A6 shows the Daubechies wavelet transform on the image at the level 3 decomposition.

CHAPTER 5
HARDWARE IMPLEMENTATION USING FPGAS

In an ideal hardware implementation, we prefer the inputs and results be integers to keep the computations simple. So ideal wavelet transform for a hardware implementation are those that map integer to integer [20]. Discrete Wavelet transform was implemented by filter banks in the software implementations. Among the two kinds of wavelet transform discussed in the software implementation section, the Cohen Daubechies Feauveau Wavelet transform is the simplest one to implement in hardware and it is also the fastest. Thus the filter used is the (2, 2) Cohen-Debuchies-Feauveau wavelet filter. Though much longer filters are common for audio data, relatively short filters are used for image and video**.**

5.1 Implementation Overview

A modified form of the Bi-orthogonal (2, 2) Cohen-Debuchies-Feaveu wavelet filter is used. If there are *2k* elements in the row or column then these produce *k* low pass and *k* high pass wavelet transform coefficients. The analysis filter equations are shown below:

High pass coefficients: $g(k) = 2.x(2k+1) - x(2k) - x(2k+2)$

Low pass coefficients : $f(k) = x(2k) + (g(k-1) + g(k))/8$

where *g(k)* is the k_{th} high pass coefficient and *f(k)* is the k_{th} low pass coefficient and *x(k)* represents the input pixel value in the k_{th} position. Figure 5-1 shows the coefficient ordering for a sample row input with 512 elements i.e. with *k=512*.

The boundary conditions are handled by symmetric extension of the coefficients as shown below:

x[2], x[1], **(x[0], x[1],…….,x[n- 1], x[n]**), x[n- 1], x[n- 2]

The synthesis filter equations are shown below:

Even samples: $x(2k) = f(k) - (g(k-1) + g(k+1))/8$

Odd samples: $x(2k+1) = (g(k) + f(k) + f(k+1))/2$

DWT in X and Y directions.

Each pixel in the input frame is represented by 8 bits, accounting for 1 pixel per memory word. These 8 bits goes into the higher 8 bits (MSB) of the 16 bit memory word. Thus each memory read process brings in one consecutive pixels of a row. Thus two clock cycles generates one value each of f and g coefficients.

0 1	……….	254 255	256 257	……….	510 511

Pixel Data

f0 f1	……….	f254 f255	g0 g1	……….	g254 g255

Coefficient Data

Figure 5-1 Coefficient ordering along x (row) direction.

These have to be written back in a Mallot ordering scheme i.e. to write back the coefficients in a way to put all the low frequency coefficients *f* ahead of the high frequency coefficients *g*. This is shown in the coefficient data in Figure 5-1. It allows progressive image transmission/reconstruction. The bulk of the ‘average’ information is ahead, followed by the minor ‘difference’ information. Although an in-place write back scheme is available, it is not implemented because after the end of one row operation the coefficients would have to be ordered again to follow Mallot ordering. Once the filter has been applied along all the rows in a stage, the same filter is applied along the columns. The column coefficients are also written in the same aforementioned Mallot ordering. The column coefficient ordering is shown in Figure 5-2.

Pixel Data
0
1
:
254
255
256
257
:
510
511

Coefficient Data
f0
f1
:
f254
f255
g0
g1
:
g254
g255

Figure 5-2 Coefficient ordering along y (col) direction.

The image is stored in the memory as a column vector. Thus to enable Mallot ordering for both row and columns, unique address generation for both the reads and the writes is required.

The current hardware implementation processes the 512 by 512 pixel input image frame with one level of wavelet transform. In this, 512 pixels of each row are used to compute 256 high pass coefficient *g* and 256 low pass coefficients *f*, as shown in figure 5-1. The coefficients are written back in a rearranged manner such that the low frequency ones are ahead of the high frequency ones. Once all the 512 rows are processed, the filters are applied in the Y direction. Figure 5-3 shows the Mallot ordering for the Level 1 of wavelet transform.

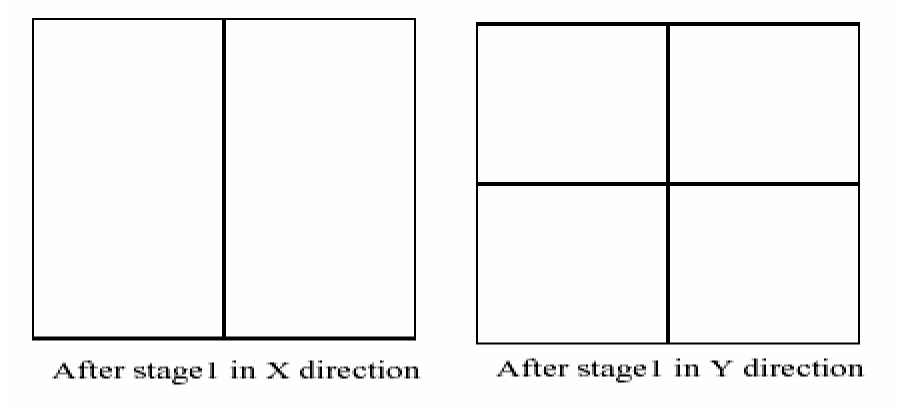


Figure 5-3 Mallot ordering along the Level 1 of wave-letting.

5.2 Implementation Scheme for DWT

The scheme involves the collaborative effort of MATLAB modules and PGM toolbox and third party software used for loading image into memory and also for loading the bit files to the FPGA on the XSV board. Figure 5-4 illustrates the implementation scheme. The 512 x 512 image in the PGM format is originally stored in the PC. It is then stripped of its image header information using the MATLAB program '*make_ram_file*'. The '*make_ram_file*' program produces a raw hex '.bit' file that is ready to be loaded into the embedded left bank ram of the board. The '*left_ram_bank.bit*' file is loaded into the FPGA using the utility GXSLOAD. This is to enable the XSVSRAM utility to write the image bit file into the left bank memory of the board. Figure B1 of Appendix B shows the screen shot of the GXSLOAD utility. Figure B2 shows the screen shot of the XSVSRAM utility. The FPGA is reconfigured again with bit file for the wavelet transform. The design uses two push buttons and one LED from the board. One of the push button controls the reset and another is configured as the start button. The wavelet transform process is commenced when the start push button on the prototyping board is pressed. The LED becomes on after the completion of the wavelet transform process. After the wavelet transform is performed the results are written into the right memory bank.

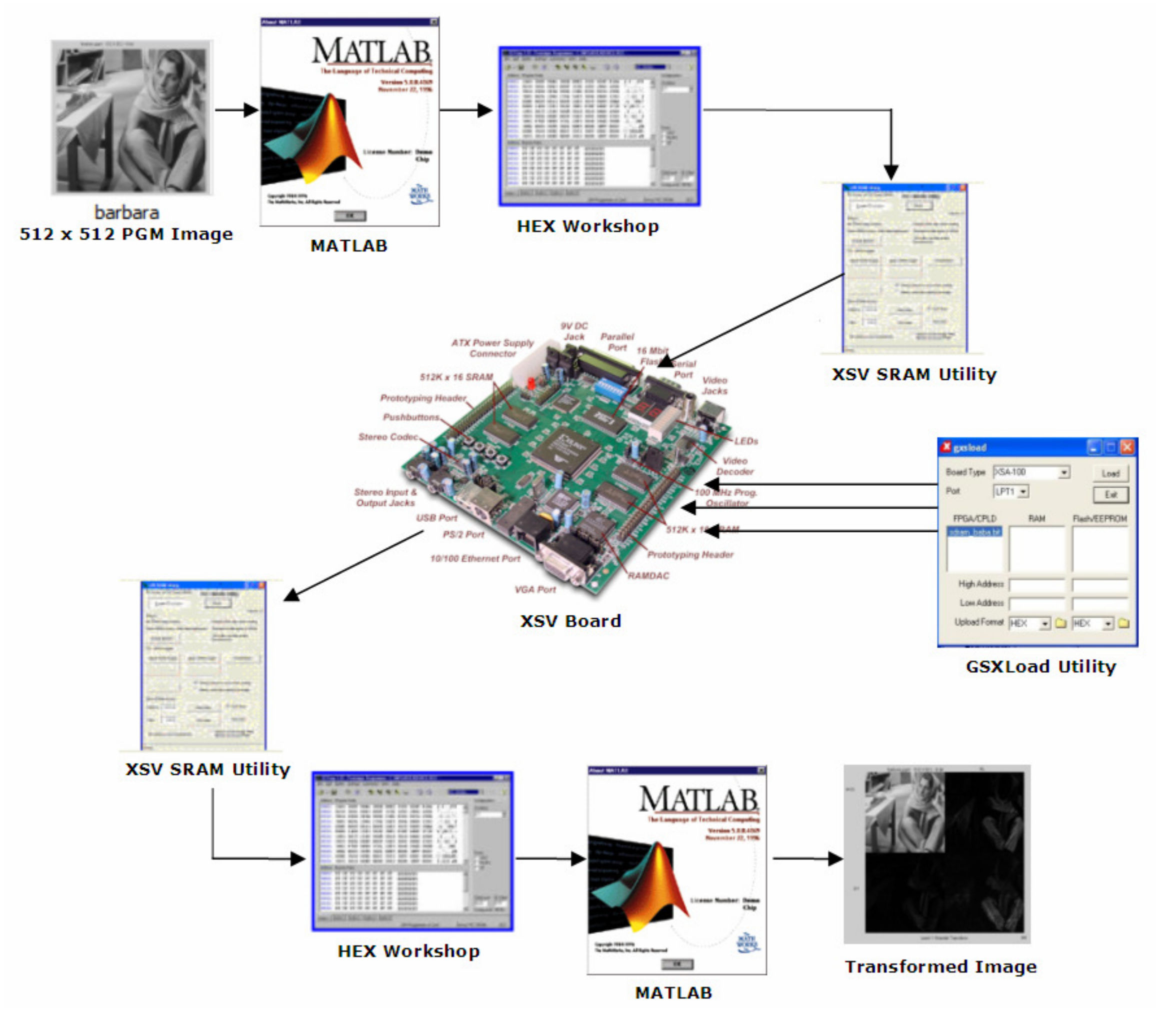


Figure 5-4 Hardware implementation scheme for DWT

The board is then configured with the bit file of the '*right_ram_bank.bit*' so that the right memory bank is read. The XSVSRAM utility then reads the right SRAM bank and transfers the memory file to hex '*.bit'* file on the PC. This hex file is converted to a PGM image file by the MATLAB Program '*makepgm512*' by adding suitable image headers. This wavelet transformed PGM image can be viewed and analyzed by the PGM viewer utility in the MATLAB PGM toolbox. Figure B3 of Appendix B shows the screen shot of the HEX WORKHOP utility that is used at both input and output stages to inspect the content of the memory files.

5.3 VHDL Modules

The design is split into three main modules. The modules are for the memory bank read, wavelet transform and for memory bank write. Thus three main VHDL modules have been coded to be able to implement this design. The coding is done in such a way that the entire code is reusable if the design is to be scaled or if it is to be fitted as sub modules for a larger design. This is done so that these modules can integrate into larger modules for entire image compression applications. Appendix C, Figure C1 shows the module hierarchy in the Xilinx ISE's Project Navigator. Both the row and column modules are almost identical. These differ only in the address generation for the memory reads and write. Figure 5-5 shows the top level view of the design in the prototyping hardware.

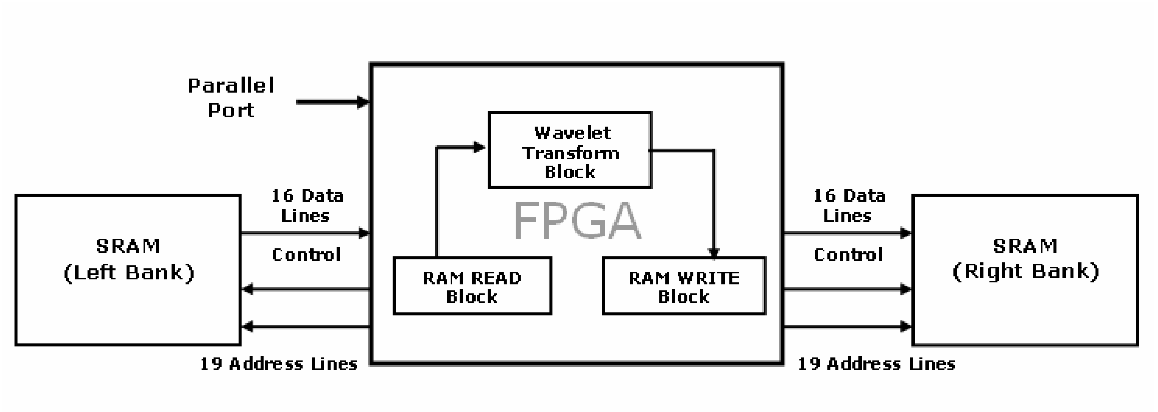


Figure 5-5 Top Level view of XSV board prototyping hardware

Top level Module

The top level module is put in as wrapper module to map the FPGA pin connections to the actual wavelet transform modules. These are 'wavelet_top_row' and 'wavelet_top_col'. The wavelet transform modules are 'wavelet_tr_row' and 'wavelet_tr_col'.

Wavelet Transform Module

This module computes the wavelet transform coefficients of the input image pixels obtained from the memory read module. After the computation, the high and low pass coefficient are passed to the memory write module. This module is also responsible for generation of the address for memory reads and writes. The module is implemented using a finite state machine (FSM) type of design. All the computations are performed using 8 bit arithmetic. Listing C1 in Appendix C lists the VHDL code for the wavelet transform showing the main FSM used for the flow.

Memory Read Module

This module accepts the read addresses generated by the wavelet transform module. A single execution of this module fetches the two image pixels for processing. This module generates the timing signal for the RAM chip select, RAM chip enable, and the RAM read signal. The data is read from the RAM and then passed on to the Wavelet transform module. The read is also implemented as a FSM. Appendix C, listing C2 lists the VHDL code for the memory read module 'lh_ram_read' showing the FSM used for the flow.

Memory Write Module

This module accepts the write addresses generated by the wavelet transform module and the coefficients generated by the wavelet transform module. A single execution of this module writes the two coefficients to the right memory bank. This module generates the timing signal for the RAM chip select, RAM chip enable, and the RAM write signal. The data is then passed on to the Wavelet transform module. The write is also implemented as a FSM. Appendix C, listing C3 lists the VHDL code for the memory write module 'rh_ram_read' showing the FSM used for the flow.

5.4 MATLAB Modules

Since Matlab did no have support to read and write PGM image format, the PGM image toolbox, ‘Portable Anymap File Format (PNM)’ Version 3.0 was installed to enable the read and write of PGM image files [21]. In addition to this, two Matlab programs *make_ram_file* and *make_pgm512* are coded to convert the PGM file to memory ‘bit’ file and vice versa. The module *make_ram_file* accepts an input PGM image file and strips of its header information and saves the file back as a raw ‘.bit’ hex memory file. The module *make_pgm512* accepts an input bit file read from the right memory bank and adds suitable headers and saves the file back as a PGM image format file.

CHAPTER 6
HARDWARE IMPLEMENTATION- RESULTS AND ANALYSIS

6.1 Simulation & Results

Before any hardware implementation and testing is performed, all the VHDL modules are tested for correct functionality using the MODELSIM functional simulation tool. The MODELSIM tool provided the necessary environment for complete functional simulation of the target hardware, i.e., Xilinx Vitex FPGAs. The MODELSIM tool features high speed and target hardware platform adaptability. It also allows for modification and verification of the VHDL codes simultaneously.

Captured outputs for a run of the wavelet transform is presented and explained here. The outputs is divided into three sections each for the memory bank read, transform process and memory bank write.

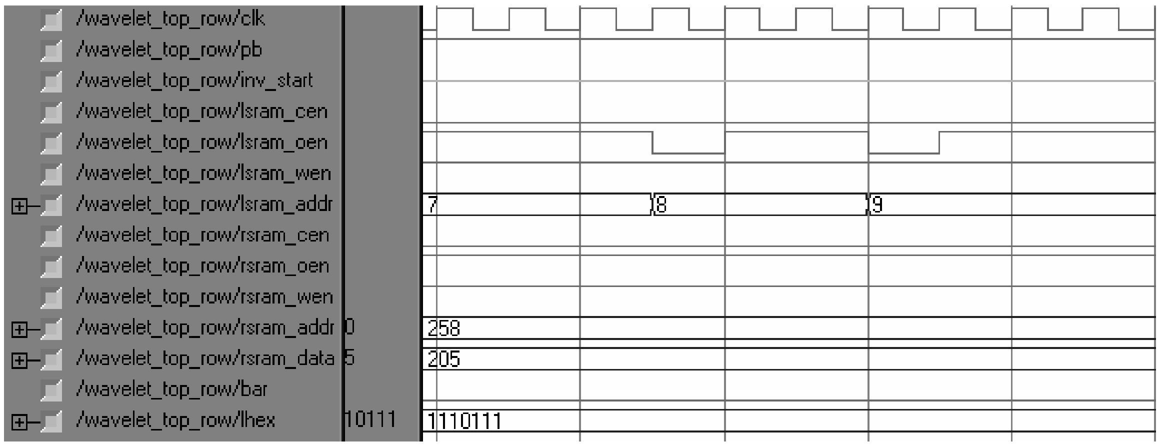

/wavelet_top_row/clk
/wavelet_top_row/pb
/wavelet_top_row/inv_start
/wavelet_top_row/lsram_cen
/wavelet_top_row/lsram_oen
/wavelet_top_row/lsram_wen
/wavelet_top_row/lsram_addr
/wavelet_top_row/rsram_cen
/wavelet_top_row/rsram_oen
/wavelet_top_row/rsram_wen
/wavelet_top_row/rsram_addr
/wavelet_top_row/rsram_data
/wavelet_top_row/bar
/wavelet_top_row/lhex
0
5
10111
7
8
9
258
205
1110111

Figure 6-1 Timing of RAM READ control signals (captured from MODELSIM Output)

Figure 6-1 captured from MODELSIM shows the timing of the ram read control signal *'lsram_oen'*. Two consecutive memory reads are performed at a time and it is seen that one read process of an image pixels takes 2 clock cycles. The signal *'lsram_addr'* holds the address of the memory location to be read from.

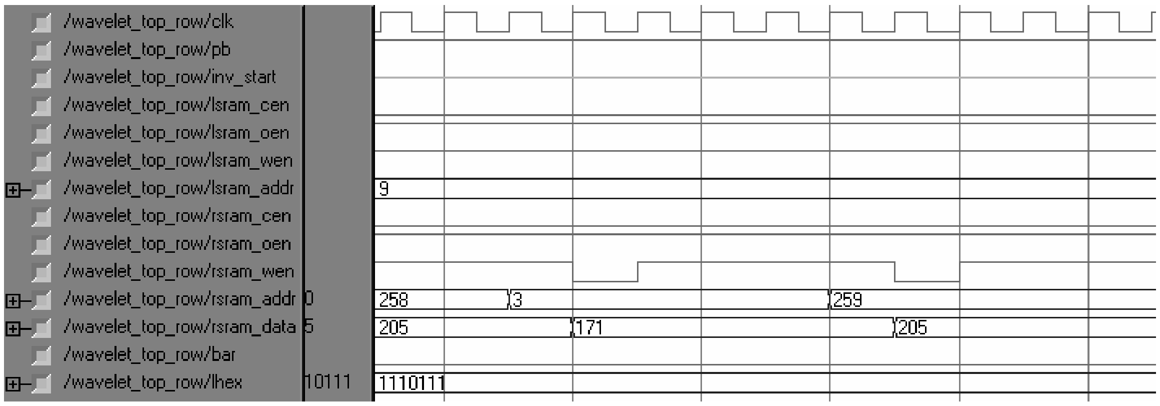

/wavelet_top_row/clk
/wavelet_top_row/pb
/wavelet_top_row/inv_start
/wavelet_top_row/lsram_cen
/wavelet_top_row/lsram_oen
/wavelet_top_row/lsram_wen
/wavelet_top_row/lsram_addr
/wavelet_top_row/rsram_cen
/wavelet_top_row/rsram_oen
/wavelet_top_row/rsram_wen
/wavelet_top_row/rsram_addr
/wavelet_top_row/rsram_data
/wavelet_top_row/bar
/wavelet_top_row/lhex
9
258
3
259
205
171
205
0
5
10111
1110111

Figure 6-2 Timing of RAM WRITE control signals (captured from MODELSIM Output)

Figure 6-2 captured from MODELSIM shows the timing of the ram write signal *'lsram_wen'*. Two different location memory writes are performed at a time and it is seen that one read process of an image pixels takes 2 clock cycles. The signal *'rsram_addr'* holds the address of the memory location to be written to.

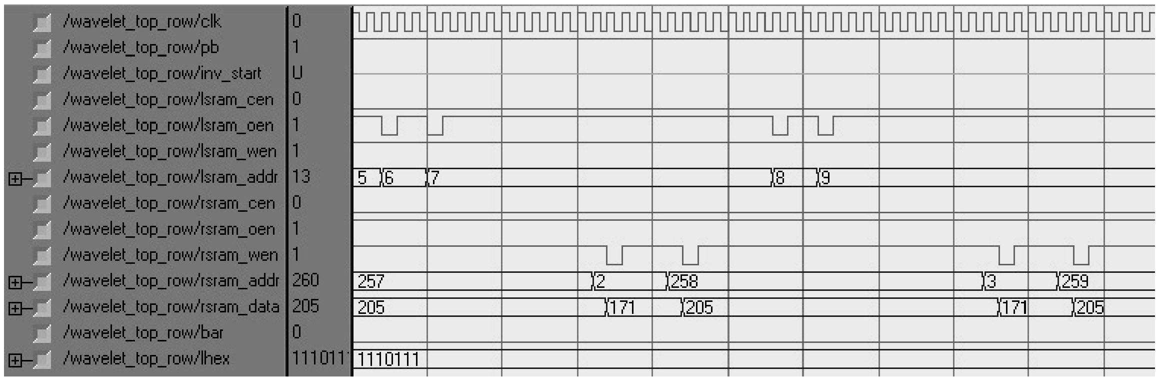

/wavelet_top_row/clk
/wavelet_top_row/pb
/wavelet_top_row/inv_start
/wavelet_top_row/lsram_cen
/wavelet_top_row/lsram_oen
/wavelet_top_row/lsram_wen
/wavelet_top_row/lsram_addr
/wavelet_top_row/rsram_cen
/wavelet_top_row/rsram_oen
/wavelet_top_row/rsram_wen
/wavelet_top_row/rsram_addr
/wavelet_top_row/rsram_data
/wavelet_top_row/bar
/wavelet_top_row/lhex
0
1
U
0
1
1
13
0
1
1
260
205
0
111011
5
6
7
8
9
257
2
258
3
259
205
171
205
171
205
1110111

Figure 6-3 Timing of the Wavelet Transform process signals (captured from MODELSIM Output)

Figures 6-3 shows the captured MODELSIM output for one complete run of the wavelet transform process on a set of two image pixels. As seen in figure 6-3 the clock cycle between the memory read and memory write is spent in the calculation of the coefficients of the wavelet transform.

6.2 Test Results

After the correct functional verification by MODELSIM, the VHDL description is synthesized using XST. A number of iterations are made to match the VHDL description to the timing and synthesis constraints. Subsequently, the implementation tool of the Xilinx ISE 6.1 is used to implement the design followed by the generation of the bit file. The hardware implementation is carried out by programming the FPGA through the parallel port of a computer. Before programming the FPGA, the image to be transformed is loaded into the left SRAM bank using the XSV SRAM Utility. The wavelet transform is started by pushing the start button on the XSV Board. After the wavelet transform is over the assigned LED on the XSV board turns on to indicate that the process is complete. The wavelet transformed image is read back from the right SRAM bank using the XSV SRAM Utility. This transformed image is viewed using MATLAB to verify the wavelet transform results.

Equivalent C and MATLAB programs are coded to evaluate the execution time taken by the software implementation. Tests are performed on six images. Table 6-1 and Figure 6-4 below shows the comparison of the speedup achieved by the hardware implementation in comparison to the software methods for images with 8 bit depth and for 1D wavelet transform. The software execution time is computed only for the image transform process and does not include the time taken for image file access, read and write back.

Table 6-1 Performance analysis on the results for software implementation of the 1D wavelet transform compared with their actual hardware implementation

	Unit			
Picture Format	Pixel	512 x 512	256 x 256	128 x 128
MATLAB Execution Time	**µsec**	9078000	3125000	1187000
C (GNU) Execution Time	**µsec**	131072	32768	8192
HW execution Time	**µsec**	6790	1691	420
Performance ration of HW Vs SW		**19. 30**	**19. 37**	**19. 50**

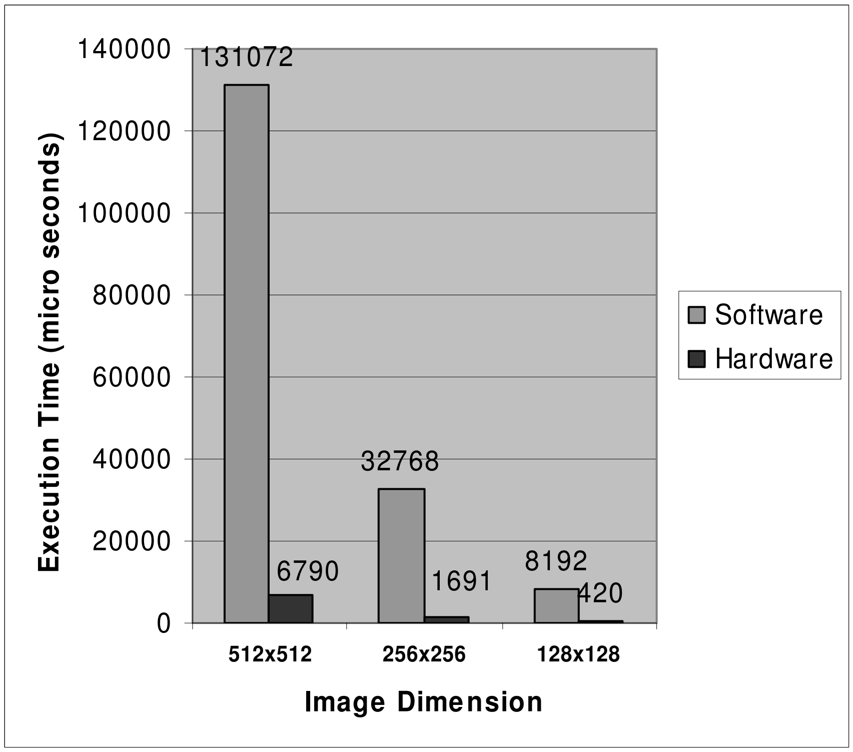


Figure 6-4 Performance comparisons between different picture sizes (for execution times) between C and Hardware.

Table 6-2 Performance analysis on the results for software implementation of the 1D wavelet transform compared with actual hardware implementation on a per pixel basis.

	Unit			
Picture Format	Pixel	512 x 512	256 x 256	128 x 128
MATLAB Execution Time	**Total Cycles/pixel**	69260.00	95367.00	144900
C (GNU) Execution Time	**Total Cycles/pixel**	1000.00	1000.00	1000.00
HW execution Time	**Total Cycles/pixel**	13.00	12.90	12.80

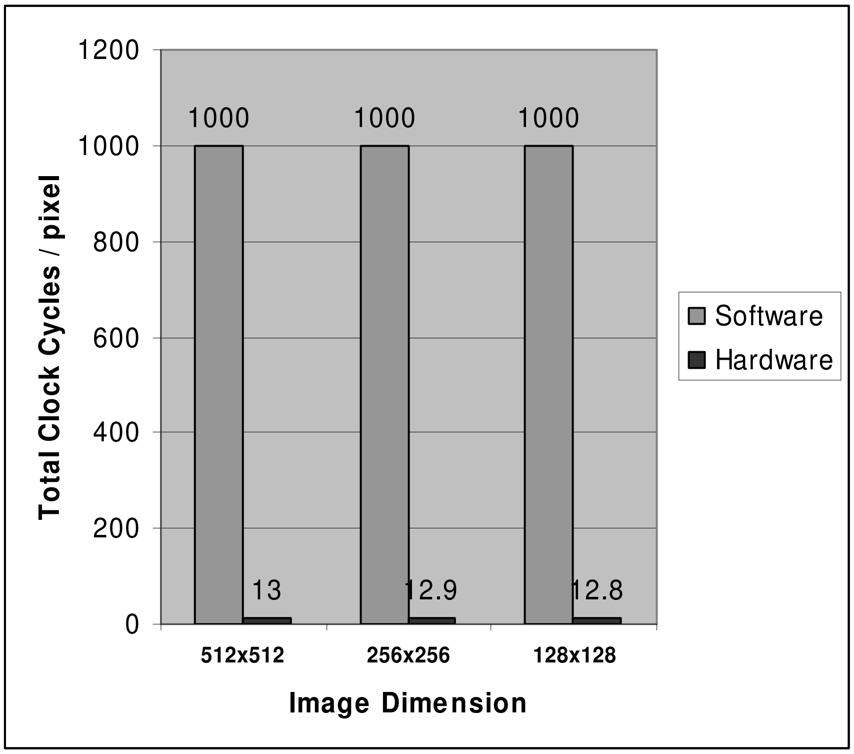


Figure 6-5 Performance comparison between different picture sizes (for effective Total clock cycles per execution) between C and Hardware.

Table 6-2 and Figure 6-5 summarizes the performance comparison calculated for the hardware and software execution on a per pixel basis. The software platform is MATLAB 6.5 and GNU C Compiler GCC Version 3.3.1 (also known as *cygming*) and the hardware used is an Intel P4 2.0 GHz PC equipped with 512MB RAM and running Windows XP Pro operating system. It is observed that the hardware speedup obtained is several times the software methods. Figures D1 and D2 of Appendix D show the wavelet transformed test images as read from the right memory bank of the FPGA Board.

6.3 Performance (Area & Speed)

Implementation results of the configurations that are used during simulation/synthesis iterations are presented here. The logic resources of the FPGA are divided in different categories as look up tables (LUT), input/output Blocks (IOB), Flip-flops, Multipliers, etc. A specific group of a number of these resources is called configurable logic blocks (CLB) and a group of CLBs forms a slice. The routing resource usage is not given by the place and route tools - higher device utilization implies greater routing resource utilization. The total number of CLBs used is a function of the device resources used and how densely they are packed. For example, on a CLB only a flip flop could be used, but still it adds to the CLB count. The usages for the three types of image dimension for 1D-WT is presented in Table 6-3.

Table 6-3 Device Usage and Timing for different image sizes.

Picture Format	LUTs (4)	**CLB (Flops)**	**Total CLB**	**Percentage Device CLB Used**	**I/O Buf**	**Timing (MHZ)**
512 x 512	663	287	665	14.14 %	78	61.52
256 x 256	663	287	665	14.14 %	78	61.18
128 x 128	663	287	665	14.14 %	78	61.52

The numbers given in resource usage above, show that Xilinx v800hq240-6 device can easily accommodate the design. Larger practical picture sizes merely require an FPGA with more internal or external RAM, i.e., the amount of required logic does not grow, as seen in Table 6-3. For cost reduction, one might want to consider using the smallest possible FPGA that can accommodate the necessary resources and providing an external dual port RAM large enough for the desired picture dimension. Since only about 14% of the FPGA is used, wavelet implementations using other filters like the Daubechies wavelet filter can also be targeted for implementation on the Xilinx v800hq240-6 FPGA.

CHAPTER 7
CONCLUSIONS AND FUTURE EXTENSIONS

7.1 Conclusions

This thesis introduced the basic wavelet theory used for wavelet transform based image compression. FPGA based hardware design methodology is presented. DWT algorithm using the *(2, 2) CDF* wavelet and *Debauchies-4 tap* wavelet is studied and implemented in software. Since the *(2, 2) CDF* based DWT is the simplest, it is selected for hardware implementation. The *(2, 2) CDF* algorithm is studied and mapped for hardware implementation. The algorithm is behaviorally described using VHDL. After successful verification the design is synthesized and tested on prototype hardware. Performance analysis of the hardware design with respect to software is evaluated. It is seen that the hardware speed up obtained is several times compared to software implementations. This framework for an FPGA-based DWT system would allow a signal/image processing application developer to generate FPGA configurations for DWT at a high level rather than spending considerable time learning and designing at a gate and routing level. Thus, the end-user will benefit from the high performances of FPGA devices while designing at a high level with familiar tools. The results are promising when compared to software; however, further work needs to be done towards the extension of the system to handle different arithmetic representation, different wavelet analysis and synthesis schemes along with different architectures.

Wavelet based image compression is ideal for adaptive compression since it is inherently a multi-resolution scheme. Variable levels of compression can be easily achieved. The number of wave-letting stages can be varied, resulting in different number of sub bands. Different filter banks with different characteristics can be used. Efficient fast algorithm (pyramidal computing scheme) for the computation of discrete wavelet coefficients makes a wavelet transform based encoder computationally efficient.

Reconfigurable hardware is best suited for rapid prototyping applications where the lead time for implementation can be critical. It is an ideal development environment, since bugs can be fixed and multiple design iterations can be done, with out incurring any non recurring engineering costs. Reconfigurable hardware is also suited for applications with rapidly changing requirements. In effect, the same piece of silicon can be reused.

With respect to limitations, achieving good timing/area performance on these FPGAs is much harder, when compared to an ASIC or a custom IC implementation. There are two reasons for this. The first pertains to the fixed size look-up tables. This leads to under utilization of the device. The second reason is that the pre-fabricated routing resources run out fast with higher device utilization.

7.2 Future Work

The lessons learned from this work will help us enhance similar implementations in the future. Few of the improvements that we now foresee are listed below:

- Build a corresponding wavelet transform decoder on the FPGA and demonstrate the adaptability of the encoder-decoder pair. The encoder would need to signal the decoder on which codec is being used.
- Data movement from host to embedded memory and back to host takes a significant amount of the processing time. Data movement could have been minimized by implementing both the stages of the encoder on a single FPGA. On the other side, when these FPGAs are utilized more than about 40%, the timing performance drops sharply. This is because it runs out of routing resources. An

alternate architecture would be to use the two FPGA Boards for the two stages (to get good timing), but use the local bus on the board to transfer data from first to the second board.

- A suggestion with respect to embedded memory architecture is to have two embedded memory chips attached to the FPGA, so that is can work as a double buffer. Here, the host can refill the next frame on one of the memory chips, while the first FPGA is still working with the other. Another option is to use the Block Select RAM present in the FPGA.

APPENDIX A

SOFTWARE IMPLEMENTATION CODE & RESULTS

Listing A1. Matlab function code for wavelet transform block in 'wavelet_t1.m'

```
% function declaration for the wavelet transform:-

function A1 = wavelet_trans(x,n,st,off1)
s=zeros(256,1);
d=zeros(256,1);
mid=0;i=0;
mid = (n/2)-1;
for i= 0:mid
    s(i+1)=x(off1+(2*i*st)+1);
    d(i+1)=x(off1+(2*i*st)+st+1);
end
d(1)=d(1)+d(1)-s(1)-s(1+1);
s(1)=s(1)+((d(1)+d(1))/8);
for i= 1:mid-1
    d(i+1)=d(i+1)+d(i+1)-s(i+1)-s(i+1+1);
    s(i+1)=s(i+1)+((d(i-1+1)+d(i+1))/8);
end
d(mid+1)=d(mid+1)+d(mid+1)-s(mid+1)-s(mid+1);
s(mid+1)=s(mid+1)+((d(mid-1+1)+d(mid+1))/8);
for i= 0:mid
    x(off1+(i*st)+1)=s(i+1);
    x(off1+((i+mid+1)*st)+1)=d(i+1);
end
A1=x;
```

Listing A2. Matlab code fragment implementing the three level wavelet transform block in 'wavelet_t1.m'

```
ROW=1;
COL=512;
A1=A;
nt=512;
for i = 0:512: (nt-1)*512
    A1=wavelet_trans(A1,nt,ROW,i);
end;
for i = 0:nt-1
```

```
    A1=wavelet_trans(A1,nt,COL,i);
end;
nt=256;
for i = 0:512: (nt-1)*512
    A1=wavelet_trans(A1,nt,ROW,i);
end;
for i = 0:nt-1
    A1=wavelet_trans(A1,nt,COL,i);
end;
nt=128;
for i = 0:512: (nt-1)*512
    A1=wavelet_trans(A1,nt,ROW,i);
end;
for i = 0:nt-1
    A1=wavelet_trans(A1,nt,COL,i);
end;

result=A1;
```

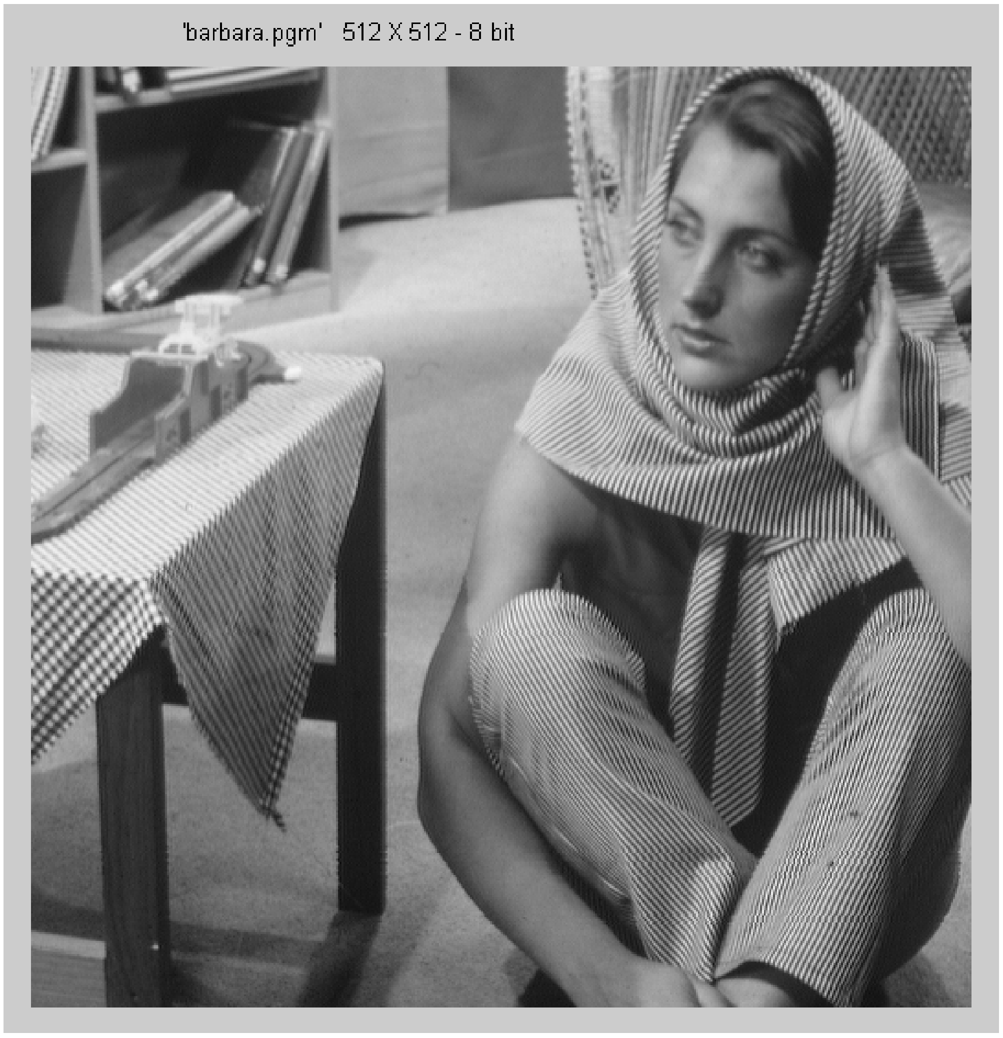


Figure A1 Original Image ‘barbara.pgm’ 512X512

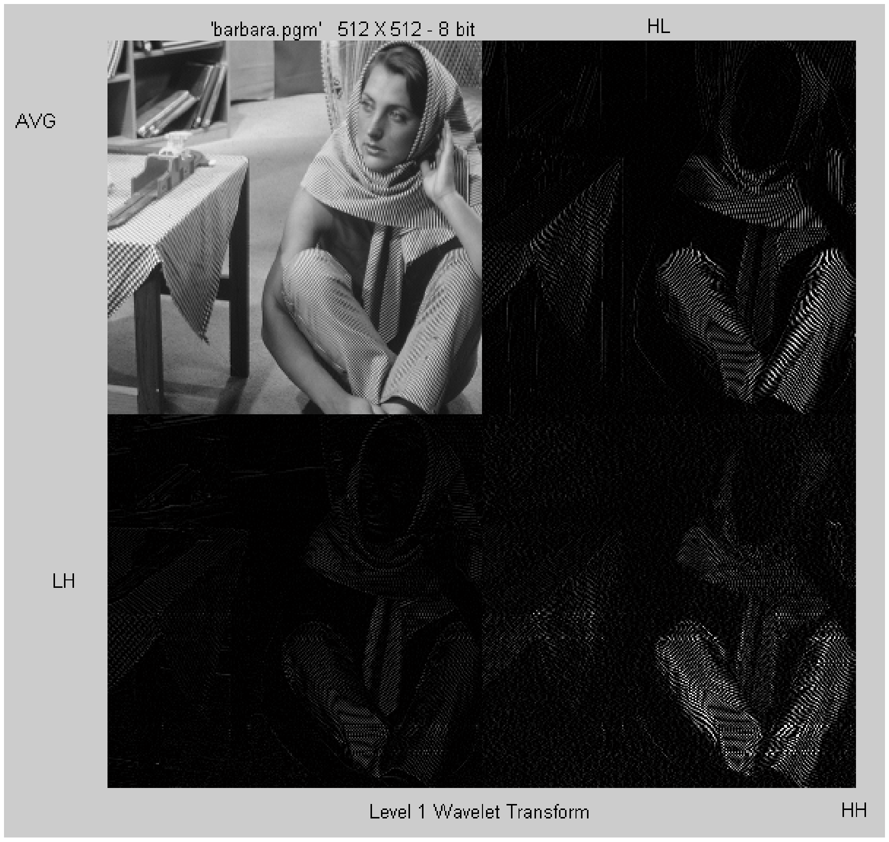


Figure A2 Wavelet Transform on ‘barbara.pgm’ at Level 1

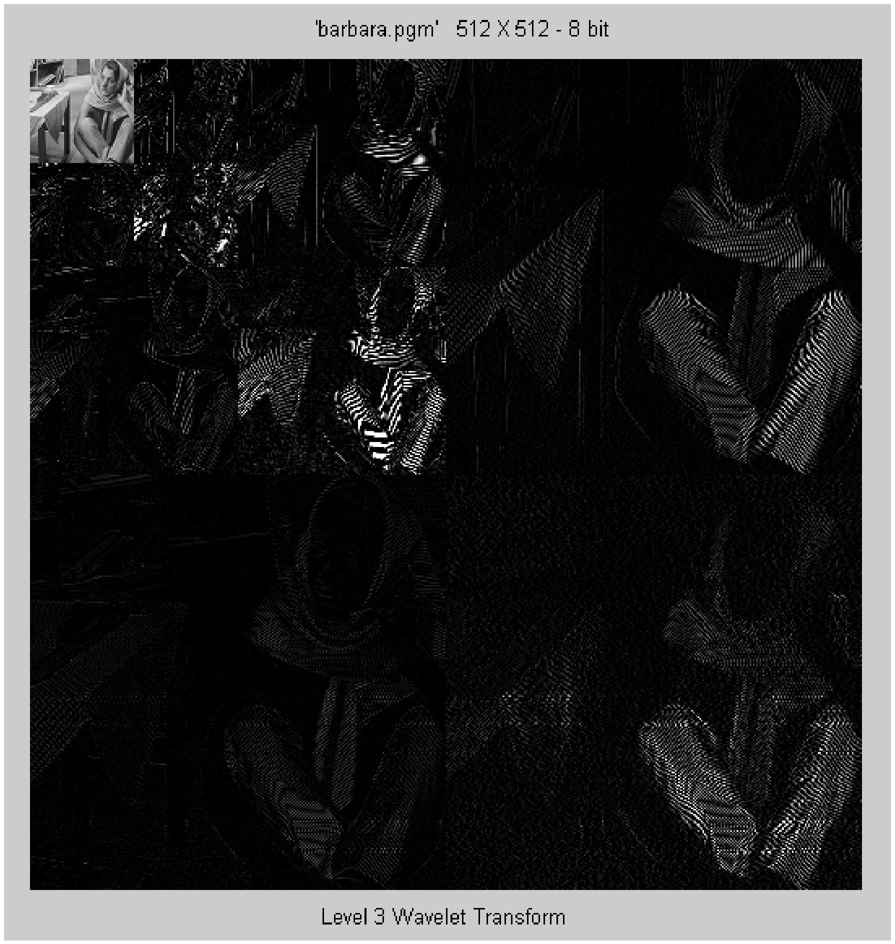


Figure A3 Wavelet Transform on 'barbara.pgm' at Level 3

Listing A3. Matlab function code for wavelet transform block in 'wavelet_db.m'

```
function A1 = wavelet_trans(x,n,st,off1)
%-----------------------------------------------------------------
%   Define filter coefficients here
%   Type of filter = Dabeuchies 4 Tap Filter
%   Averaging filter coeffcients
h0=.4829629131445341;
h1=.8365163037378077;
h2=.2241438680420134;
h3=-.1294095225512603;
h4=0;
h5=0;
%   Detail filter coeffcients
g0=h3;
g1=-h2;
g2=h1;
g3=-h0;
g4=0;
g5=0;
```

```
%   Number of Taps
T=4;

%----------------------------------------------------------------------
%line is extended by 4 places to account for boundary
line=zeros(n+4,1);
s=zeros(n/2,1);
d=zeros(n/2,1);
mid=0;i=0;
mid = (n/2)-1;
for i= 0:n-1
    line(i+1)=x(off1+i*st+1);
end
for i= n+1:n+4
    line(i)=x(n);   %pad with the last coefficient
end

z=1;
for i=1:2:(n-1)
  s(z)=line(i)*h0 + line(i+1)*h1 + line(i+2)*h2+ line(i+3)*h3 +
line(i+4)*h4 + line(i+5)*h5;
  d(z)=line(i)*g0 + line(i+1)*g1 + line(i+2)*g2+ line(i+3)*g3 +
line(i+4)*g4 + line(i+5)*g5;
  z=z+1;
end

for i= 0:mid
    x(off1+(i*st)+1)=s(i+1);
    x(off1+((i+mid+1)*st)+1)=d(i+1);
end
A1=x;
```

Listing A4. Matlab code implementing the three level wavelet transform block in ‘wavelet_db.m’

```
ROW=1;
COL=512;
D=512; % size fo the image
A=zeros(D*D,1);
k=1;
load boat.mat;
result=boat_img;
for r=1:D
    for s=1:D
        A(k)=boat_img(r,s);
        k=k+1;
    end
end
A1=A;
nt=512;
% for row operation
for i = 0:D: (nt-1)*D
```

```
   A1=wavelet_trans(A1,nt,ROW,i);
end;
% for column operation
for i = 0:nt-1
    A1=wavelet_trans(A1,nt,COL,i);
end;
nt=256;
% for row operation
for i = 0:D: (nt-1)*D
   A1=wavelet_trans(A1,nt,ROW,i);
end;
% for column operation
for i = 0:nt-1
    A1=wavelet_trans(A1,nt,COL,i);
end;
nt=128;
% for row operation
for i = 0:D: (nt-1)*D
   A1=wavelet_trans(A1,nt,ROW,i);
end;
% for column operation
for i = 0:nt-1
    A1=wavelet_trans(A1,nt,COL,i);
end;
k=1;
for r=1:D
    for s=1:D
        result(r,s)=uint8(A1(k));
        k=k+1;
    end
end
imshow(result);
```

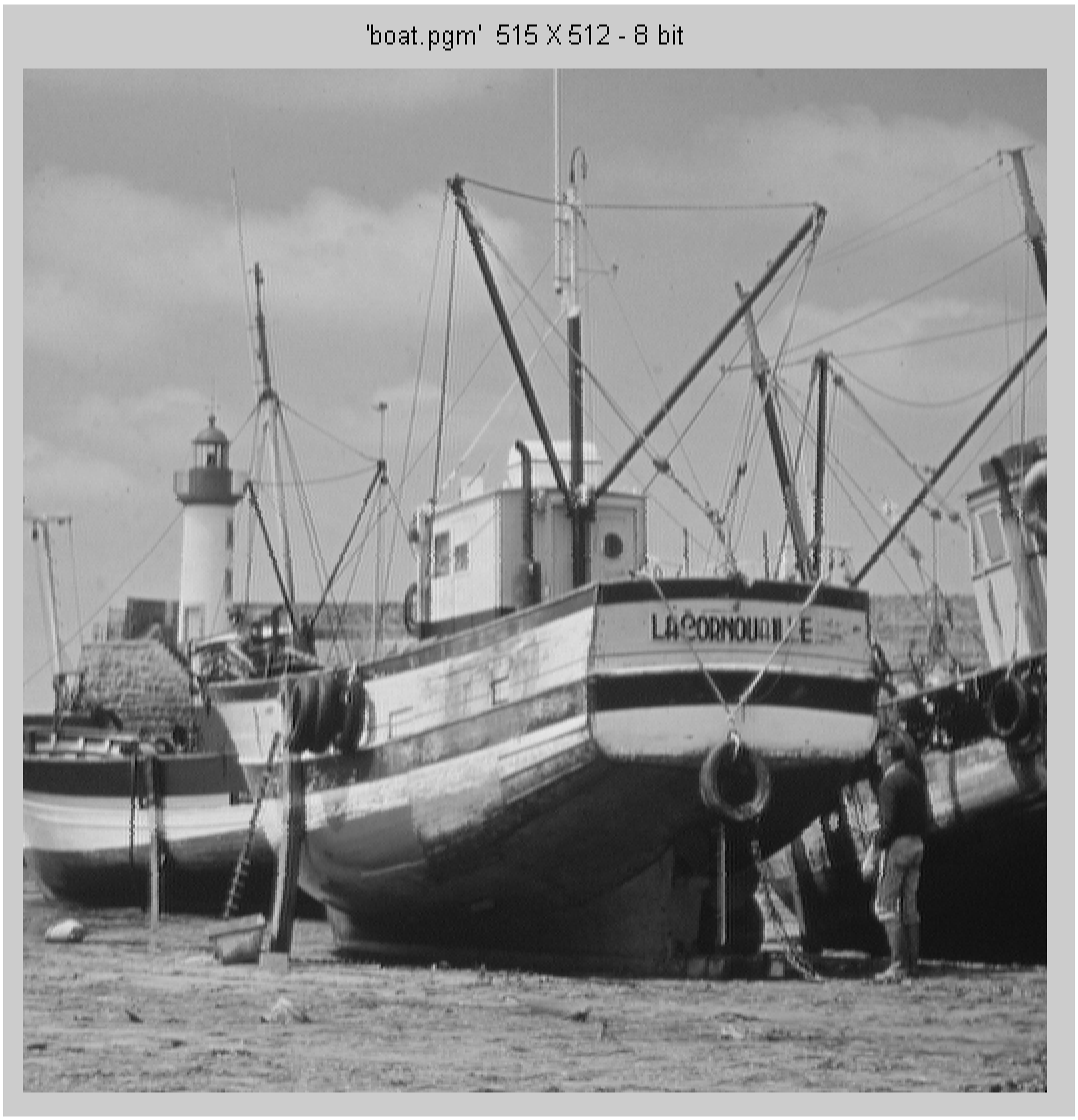


Figure A4 Original Image ‘boat.pgm’ 512X512

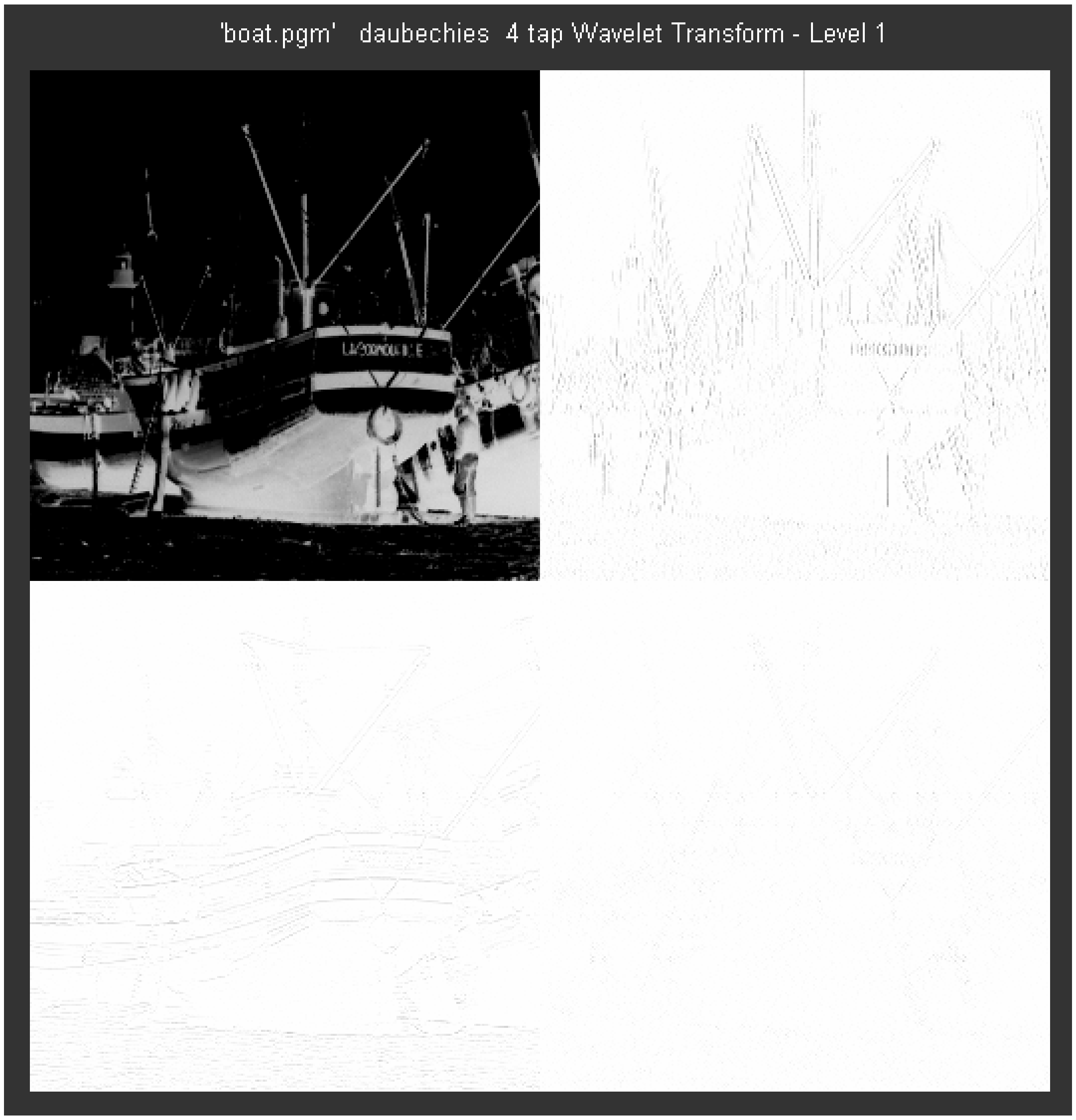


Figure A5 Wavelet Transform (Daubechies 4 Tap) on ‘barbara.pgm’ at Level 1

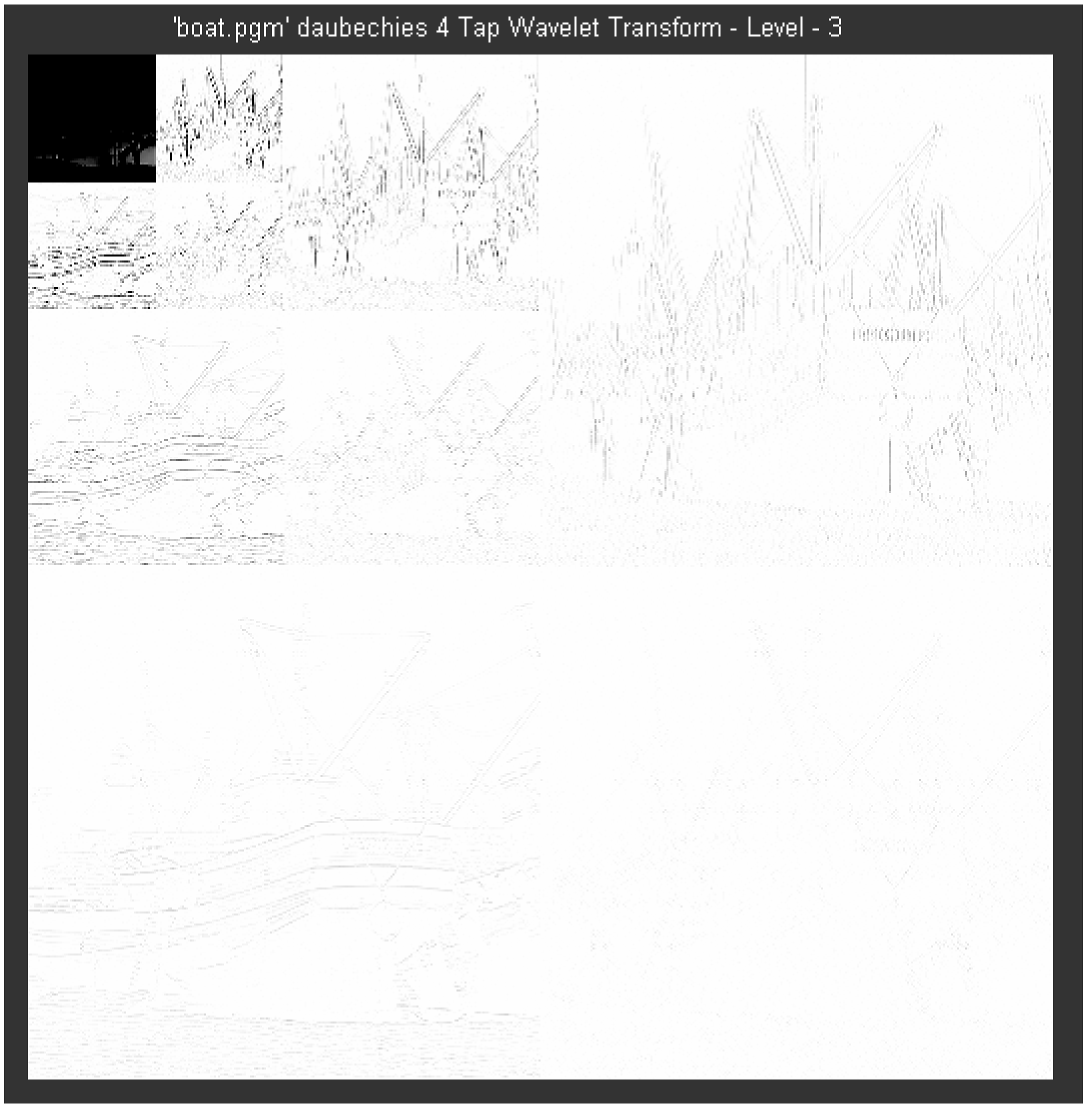
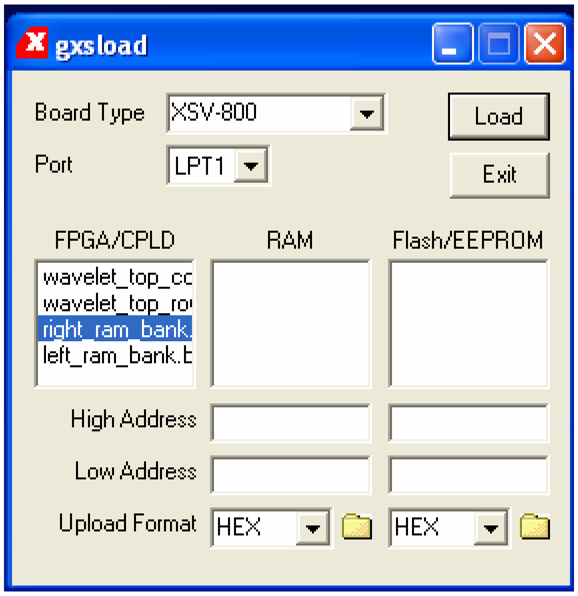


Figure A6 Wavelet Transform (Daubechies 4 Tap) on ‘barbara.pgm’ at Level 3

APPENDIX B

SCREEN SHOT OF SOFTWARE TOOLS

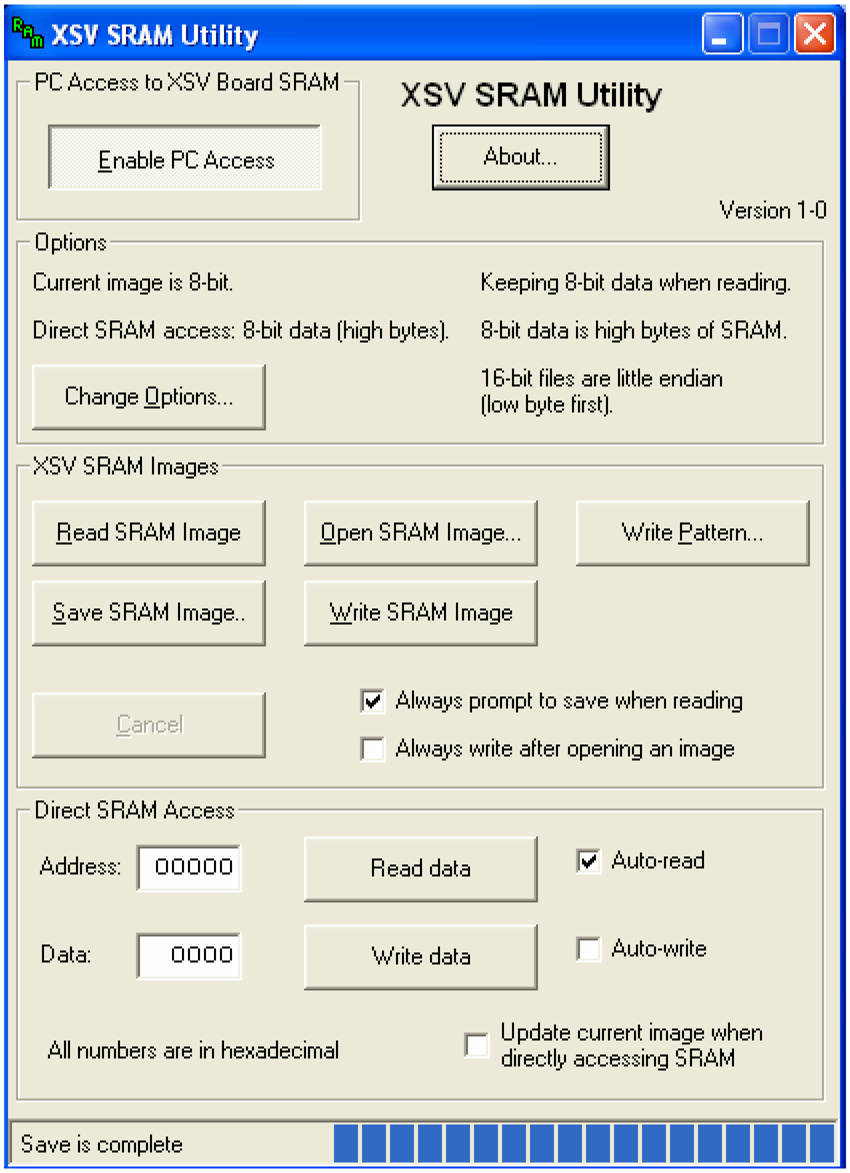


Board Type XSV-800
Port LPT1

wavelet_top_cc
wavelet_top_ro
right_ram_bank.
left_ram_bank.b

Upload Format HEX HEX

Figure B1 Screen Shot of GXSLOAD Utility.


XSV SRAM Utility

PC Access to XSV Board SRAM

XSV SRAM Utility

Version 1-0

Options

Current image is 8-bit.

Keeping 8-bit data when reading.

Direct SRAM access: 8-bit data (high bytes).

8-bit data is high bytes of SRAM.

16-bit files are little endian (low byte first).

XSV SRAM Images

[✓] Always prompt to save when reading

[ ] Always write after opening an image

Direct SRAM Access

Address: 00000

[✓] Auto-read

Data: 0000

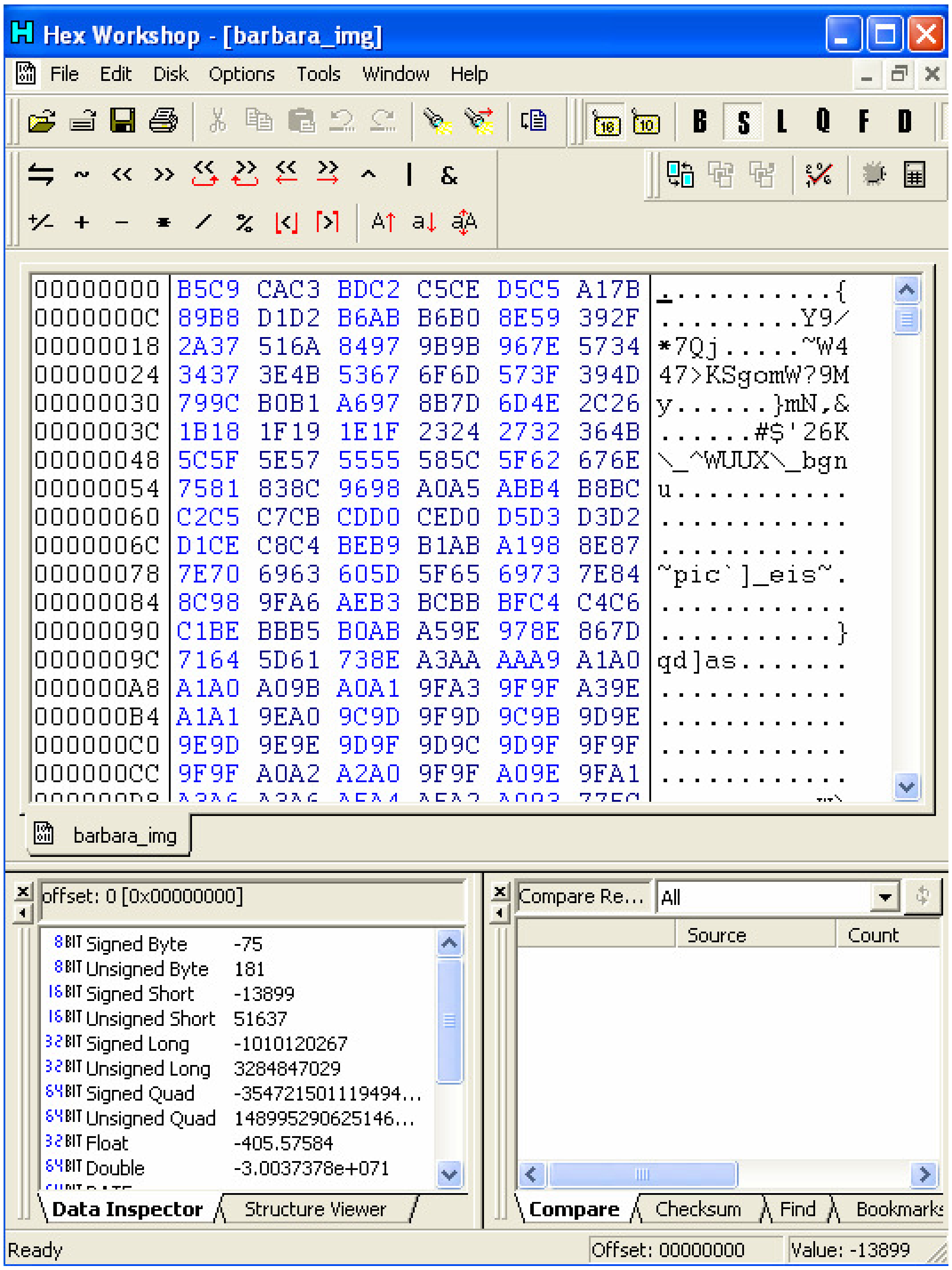
[ ] Auto-write

All numbers are in hexadecimal

[ ] Update current image when directly accessing SRAM

Save is complete

Figure B2 Screen Shot of XSVSRAM Utility


Hex Workshop - [barbara_img]
File Edit Disk Options Tools Window Help
barbara_img
Compare Re... All
Source Count
Data Inspector Structure Viewer
Compare Checksum Find Bookmarks

```
00000000 B5C9 CAC3 BDC2 C5CE D5C5 A17B ...........{
0000000C 89B8 D1D2 B6AB B6B0 8E59 392F .........Y9/
00000018 2A37 516A 8497 9B9B 967E 5734 *7Qj.....~W4
00000024 3437 3E4B 5367 6F6D 573F 394D 47>KSgomW?9M
00000030 799C B0B1 A697 8B7D 6D4E 2C26 y......}mN,&
0000003C 1B18 1F19 1E1F 2324 2732 364B ......#$'26K
00000048 5C5F 5E57 5555 585C 5F62 676E \_^WUUX\_bgn
00000054 7581 838C 9698 A0A5 ABB4 B8BC u...........
00000060 C2C5 C7CB CDD0 CED0 D5D3 D3D2 ............
0000006C D1CE C8C4 BEB9 B1AB A198 8E87 ............
00000078 7E70 6963 605D 5F65 6973 7E84 ~pic`]_eis~.
00000084 8C98 9FA6 AEB3 BCBB BFC4 C4C6 ............
00000090 C1BE BBB5 B0AB A59E 978E 867D ...........}
0000009C 7164 5D61 738E A3AA AAA9 A1A0 qd]as.......
000000A8 A1A0 A09B A0A1 9FA3 9F9F A39E ............
000000B4 A1A1 9EA0 9C9D 9F9D 9C9B 9D9E ............
000000C0 9E9D 9E9E 9D9F 9D9C 9D9F 9F9F ............
000000CC 9F9F A0A2 A2A0 9F9F A09E 9FA1 ............
```

offset: 0 [0x00000000]

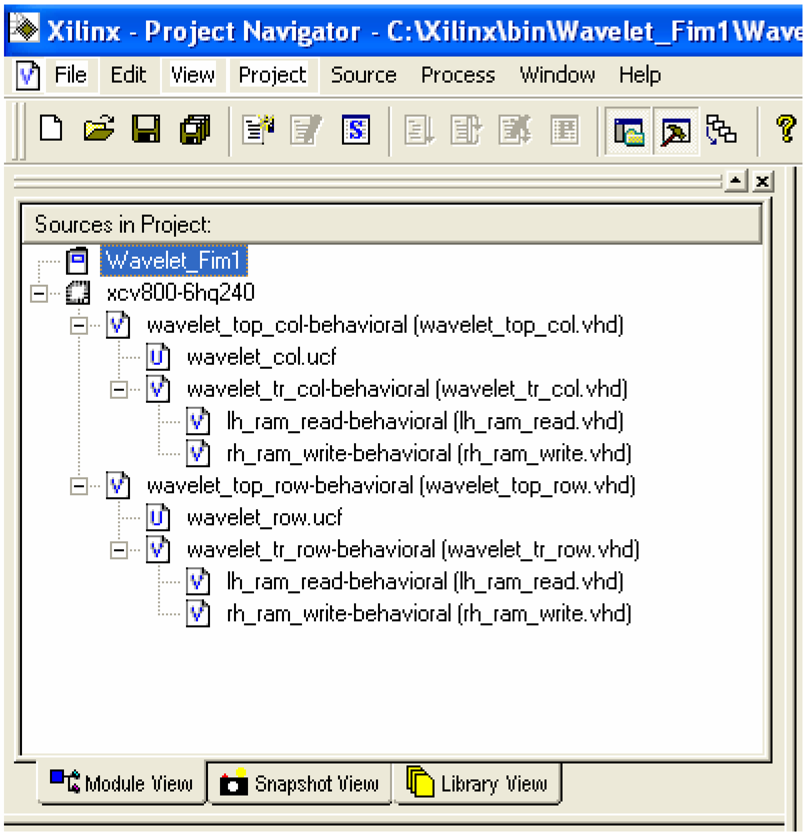
8 BIT Signed Byte	-75
8 BIT Unsigned Byte	181
16 BIT Signed Short	-13899
16 BIT Unsigned Short	51637
32 BIT Signed Long	-1010120267
32 BIT Unsigned Long	3284847029
64 BIT Signed Quad	-354721501119494...
64 BIT Unsigned Quad	148995290625146...
32 BIT Float	-405.57584
64 BIT Double	-3.0037378e+071

Ready Offset: 00000000 Value: -13899

Figure B3 Screen Shot of HexWorksop

APPENDIX C

VHDL CODE OF THE WAVELET TRANSFORM MODULES



Sources in Project:

```
Wavelet_Fim1
xcv800-6hq240
    wavelet_top_col-behavioral (wavelet_top_col.vhd)
        wavelet_col.ucf
        wavelet_tr_col-behavioral (wavelet_tr_col.vhd)
            lh_ram_read-behavioral (lh_ram_read.vhd)
            rh_ram_write-behavioral (rh_ram_write.vhd)
    wavelet_top_row-behavioral (wavelet_top_row.vhd)
        wavelet_row.ucf
        wavelet_tr_row-behavioral (wavelet_tr_row.vhd)
            lh_ram_read-behavioral (lh_ram_read.vhd)
            rh_ram_write-behavioral (rh_ram_write.vhd)
```

Figure C1 Screen Shot of Project Module View in XILINX ISE

Listing C1. VHDL code fragment for the wavelet transform module 'wavelet_tr_row' showing the main FSM controlling the flow.

```
P1:
PROCESS(state_main,fcd_process1_completed,fcd_process3_completed,start,
reset,j_row)
      variable d_var   : STD_LOGIC_VECTOR(7 downto 0);
      variable d_var1  : STD_LOGIC_VECTOR(7 downto 0);
      begin
         -- default values of signals
            --
            SIG_LHEX <= "1110111";
            inc_r <= '0';
            inc_c <= '0';

            clr_r <= '0';
            clr_c <= '0';

            inc_ivar16 <= '0';
            inc_ivar1 <= '0';
            clr_ivar  <= '0';

            fcd_process1 <= '0';
            fcd_process3 <= '0';

            clr_i1 <= '0';
            inc_i1 <= '0';

            do_latch0 <='0';
            do_latch1 <='0';
            do_latch2 <='0';

            do_latch3f <='0';
            do_latch3m <='0';

            do_latch4f <='0';
            do_latch4m <='0';
            do_latch4l <='0';

            do_latch5f <='0';
            do_latch5m <='0';
            do_latch5l <='0';
      case state_main is
            when ST_RESET =>
                        -- wait for start button to be pushed
                        SIG_BAR <= '0';
                        if start = '0' then
                              state_main_next <= ST_RESET;
                        else
                              state_main_next <= ROW_MAIN;
```

```
          end if;
when ROW_MAIN =>
          clr_ivar <= '1';
          clr_r <= '1';
          state_main_next <= ROW_CLEAR;
when ROW_CLEAR =>
       -- For Row Operation
          clr_i1 <= '1';
          state_main_next <= R_READ_0;
          -- read zeroth element
when R_READ_0 =>
          fcd_process1 <= '1';
          state_main_next <= R_READ_0_WAIT;
when R_READ_0_WAIT =>
          fcd_process1 <= '0';
          if fcd_process1_completed = '1' then
             do_latch0 <='1';  --to read the zeroth
                                 element
             inc_i1 <='1';  --advance to the next element
                                       i1=1
             state_main_next <= R_READ_1;
          else
                state_main_next <= R_READ_0_WAIT;
          end if;
    -- read first element
when R_READ_1 =>
          fcd_process1 <= '1';
          state_main_next <= R_READ_1_WAIT;
when R_READ_1_WAIT =>
          fcd_process1 <= '0';
          if fcd_process1_completed = '1' then
             do_latch1 <='1';  --to read the first
                                 element
             --inc_i1 <='1';  --advance to the next
                                 element
             state_main_next <= R_fcd_first_main1;
          else
                state_main_next <= R_READ_1_WAIT;
          end if;
when R_fcd_first_main1 =>
           d_rf_i <= (d_c_i + d_c_i) - (s_c_i + s_n_i);
           do_latch5f <='1';        -- to assign d_r to
                      d_r_c and writeback for d_r
           state_main_next <= R_fcd_first_main1A;
when R_fcd_first_main1A =>
    s_rf_i <= s_c_i + (d_r_c_i + d_r_c_i)/8;
          do_latch4f <='1';    -- to assign write back
                      for s_r
          state_main_next <= R_WRITE1;
when R_WRITE1 =>
            fcd_process3 <= '1';
            do_latch2 <='1';        -- to move next input
                      to current input
            state_main_next<=R_WRITE_WAIT1;
when R_WRITE_WAIT1 =>
          fcd_process3 <= '0';
          if fcd_process3_completed = '1'     then
```

```
                  do_latch3f <='1';    -- to assign previous
                        d to result
                  state_main_next<=R_READ_MAIN;
                  inc_i1 <='1';  --advance to i1=2
            else
                  state_main_next<=R_WRITE_WAIT1;
            end if;
  when R_READ_MAIN =>
            fcd_process1 <= '1';
            state_main_next <= R_READ_MAIN_WAIT;
  when R_READ_MAIN_WAIT =>
            fcd_process1 <= '0';
            if fcd_process1_completed = '1' then
               do_latch1 <='1';  --to read the first nth
                                     element
               state_main_next <= R_fcd_main;
            else
                  state_main_next <= R_READ_MAIN_WAIT;
            end if;
  when R_fcd_main =>
            d_rm_i <= (d_c_i + d_c_i) - (s_c_i + s_n_i);
      do_latch5m <='1';    -- to assign d_r to d_r_c and
                        writeback for d_r
            state_main_next <= R_fcd_mainA;
  when R_fcd_mainA =>
            s_rm_i <= s_c_i + (d_r_p_i + d_r_c_i)/8;
            do_latch4m <='1';     -- to assign write back
                        for s_r
            state_main_next <= R_MAIN_WRITE;
  when R_MAIN_WRITE =>
            fcd_process3 <= '1';
            do_latch2 <='1';     -- to move next input to
                        current input
            state_main_next<=R_MAIN_WRITE_WAIT;
  when R_MAIN_WRITE_WAIT =>
            fcd_process3 <= '0';
            if fcd_process3_completed = '1'      then
                  state_main_next<=R_MAIN_COUNT;  --go back
                              to read for process main
                  do_latch3m <='1';    -- to assign previous
                              d to result
                  inc_i1 <='1';  --advance to i1=2
            else
                  state_main_next<=R_MAIN_WRITE_WAIT;
            end if;
       when R_MAIN_COUNT =>
                  if i1 = mid then
                  state_main_next <= R_fcd_last; ---to go
                  process for the last element
            else
                  state_main_next <= R_READ_MAIN;
            end if;
       when R_fcd_last =>
            d_rl_i <= d_c_i + d_c_i -s_c_i - s_c_i;
      do_latch5l <='1';    -- to assign d_r to d_r_c and
                  writeback for d_r
            state_main_next <= R_fcd_lastA;
```

```
                  when R_fcd_lastA =>
                        s_rl_i <= s_c_i + (d_r_p_i + d_r_c_i)/8;
                        do_latch4l <='1';     -- to assign write back
                        inc_i1 <='1';  -- for the last element
                                          needs to be put here
                        state_main_next <= R_fcd_last_write;
             when R_fcd_last_write =>
                          fcd_process3 <= '1';
                          state_main_next<=R_fcd_last_write_wait;
             when R_fcd_last_write_wait =>
                              fcd_process3 <= '0';
                              if fcd_process3_completed = '1'     then
                                    state_main_next<=R_DONE;        --
                                          one row is done now
                              else
                              state_main_next<=R_fcd_last_write_wait;
                              end if;
             when R_DONE =>
                     --For Row Operation
                         inc_ivar16 <='1';
                         inc_r <= '1';
                         state_main_next <= R_END;
             when R_END =>
                  -- For Row Operation
                        if j_row = c512 then
                              state_main_next <= DONE;       --all rows
                              are done
                        else
                              state_main_next <= ROW_CLEAR;  --go to
                              process next row
                        end if;
            when DONE =>
                        SIG_BAR <= '1';
                        SIG_LHEX <=   "1111111";
                        state_main_next <= DONE;
      end case;
end process P1;
```

Listing C2. VHDL code fragment for the module 'lh_ram_read' showing the main FSM

```
P4: process(clk,fcd_process1,reset)   ----RAM Read Process from Left
Bank
begin
  -- read to s , d array from SRAM
 if (reset ='0') then
    RAM_READ_STATE <= START_READ;
 elsif clk'event and clk = '1' then
      Lsram_OEn <= '1';
      Lsram_WEn <= '1';
      fcd_process1_completed <= '0';
      case RAM_READ_STATE is
```

```
          when START_READ =>
             if fcd_process1 = '1' then
                  RAM_READ_STATE <=SIGNAL_S;
                else
                  RAM_READ_STATE <=START_READ;
                end if;
           when START_S =>
                  RAM_READ_STATE <=WAIT_1;
           when WAIT_1 =>
                  RAM_READ_STATE <=WAIT_2;
            when WAIT_2 =>
                  RAM_READ_STATE <=SIGNAL_S;
            when SIGNAL_S =>
                  Lsram_OEn <= '0';
            rAddress <=CONV_STD_LOGIC_VECTOR(readAddress_tempA,19);
                  RAM_READ_STATE <=READ_S;
            when READ_S =>
                  s_main16 <= Lsram_Data;
                  RAM_READ_STATE <=START_D;
            when START_D =>
                  RAM_READ_STATE <=SIGNAL_D;
            when SIGNAL_D =>
                  Lsram_OEn <= '0';
            rAddress  <=CONV_STD_LOGIC_VECTOR(readAddress_tempB,19);
                  RAM_READ_STATE <=READ_D;
            when READ_D =>
                  d_main16 <= Lsram_Data;
                  RAM_READ_STATE <=ADD_I1;
            when END_READ =>
                  fcd_process1_completed <='1';
                  RAM_READ_STATE <= START_READ;
         end case;
      end if;
end process P4;
```

Listing C3. VHDL Code fragment for the module 'lh_ram_write' showing the main FSM

```
P5: process(clk,fcd_process3,reset)   ----RAM Write Process to the
Right Bank
begin
 -- write to SRAM from s,d array
   if (reset ='0') then
      RAM_WRITE_STATE <= START_WRITE;
   elsif clk'event and clk = '1' then
      Rsram_OEn <= '1';
      Rsram_WEn <= '1';
      fcd_process3_completed <= '0';
      case RAM_WRITE_STATE is
         when START_WRITE =>
            if fcd_process3 = '1' then
                  RAM_WRITE_STATE <=START_S;
            else
```

```
                  RAM_WRITE_STATE <=START_WRITE;
            end if;
         when START_S =>
             RAM_WRITE_STATE <=WAIT_1;
         when WAIT_1 =>
             RAM_WRITE_STATE <=WAIT_2;
         when WAIT_2 =>
             RAM_WRITE_STATE <=SIGNAL_S;
         when SIGNAL_S =>
             wAddress  <= CONV_STD_LOGIC_VECTOR(writeAddress_tempA,19);
             RAM_WRITE_STATE <=WRITE_S;
         when WRITE_S =>
             Rsram_WEn <= '0';
             Rsram_Data <= s_main16R;
             RAM_WRITE_STATE <=START_D;
         when START_D =>
             Rsram_WEn <= '1';
             RAM_WRITE_STATE <=WAIT_3;
         when WAIT_3=>
             RAM_WRITE_STATE <=WAIT_4;
         when WAIT_4=>
             RAM_WRITE_STATE <=SIGNAL_D;
         when SIGNAL_D =>
             Rsram_WEn <= '1';
             wAddress  <= CONV_STD_LOGIC_VECTOR(writeAddress_tempB,19);
             RAM_WRITE_STATE <=WRITE_D;
         when WRITE_D =>
             Rsram_WEn <= '0';
             Rsram_Data <= d_main16R;
             RAM_WRITE_STATE <=ADD_I3;
         when     ADD_I3 =>
            Rsram_WEn <= '1';
            RAM_WRITE_STATE <=END_WRITE;
         when END_WRITE =>
            fcd_process3_completed <='1';
            RAM_WRITE_STATE <=START_WRITE;
      end case;
      end if;
end process P5;
```

APPENDIX D

HARDWARE TEST RESULT SCREENSHOTS

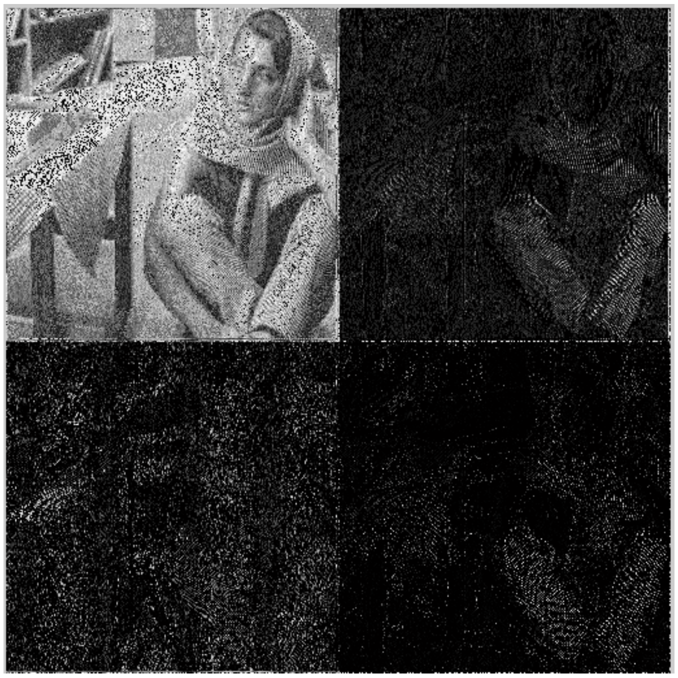


Figure D1 Screen Shot of Wavelet Transformed 512X512 Image ‘barbara’

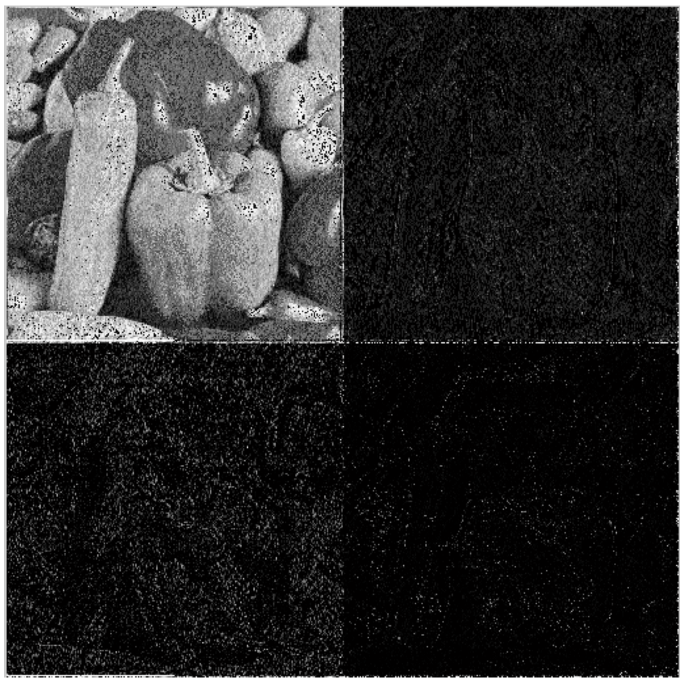


Figure D2 Screen Shot of Wavelet Transformed 512X512 Image ‘peppers’

REFERENCES

[1] M.J. Sebastian Smith, Application-Specific Integrated Circuits, Addison Wesley Longman.

[2] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, New York: 2001, p. 61-77.

[3] Xilinx Corporation. "Virtex^{TM} 2.5 Volt Field Programmable Gate Arrays," DS003-2 (v2.8.1), 2002.

[4] How much information? 2003 By UC Berkeley's School of Information Management and Systems http://www.sims.berkeley.edu/research/projects/how-much-info-2003/

[5] S. Mallat, "Multifrequency channel decompositions of images and wavelet models", IEEE Trans. Acoust., Speech, Signal Process., vol. 37, no. 12, pp. 2091-2110, Dec. 1989

[6] Peter J. Ashenden. The Designer's Guide to VHDL. Morgan Kaufmann, 1996.

[7] A. Cohen, I. Daubechies, J. Feauveau, "Biorthogonal Bases of Compactly Supported Wavelets", Communications of Pure Applied Math, vol 45, 1992.

[8] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," IEEE Trans. Signal Processing, vol. 41, pp. 3445-3462, Dec. 1993.

[9] Luiz Pires and Deepa Pandalai, Honeywell Technology Center, Minnepoli, MN 55412, "compress.c, " October 1988.

[10] Xess Co., XSV Board User Manual, http://www.xess.com/manuals/xsv-manual-v1_1.pdf

[11] D. Patterson and J. Hennessy, Computer Organization & Design: The Hardware / Software Interface, Morgan Kaufmann, 1994.

[12] Xilinx Inc., "*Virtex-II 1.5V Field-Programmable Gate Arrays*".

[13] Xilinx Co., XST User Guide, http://toolbox.xilinx.com/docsan/xilinx6/books/docs/xst/xst.pdf.

[14] “JPEG2000 Image Coding System”, JPEG 2000 final committee draft version 1.0, March 2000 (available from http://www.jpeg.org/public/fcd15444-1.pdf)

[16] Alliance 5V/3.3V 512K × 8 CMOS SRAM Specifications Sheet version 1.8 March 2002, Allianc Semiconductor.

[17] G. Strang and T. Nguyen, Wavelets and Filter Banks, Wellesley, MA:1997, p. 2, 350.

[18] M. Antonini, M. Barlaud, P.Mathieu, and I. Daubechies, "Image coding using wavelet transform," IEEE Trans. Image Processing, vol. 1, pp. 205-220, Apr. 1992.

[19] Xilinx’s glossary page. http://www.xilinx.com/support/techsup/journals/implmnt/glossary.htm..

[20] R. Calderbank and I. Daubechies and W. Sweldens and B.L. Yeo, Losless Image Compression using Integer to Integer Wavelet Transforms,International Conference on Image Processing (ICIP), Vol. I, 1997.

[21] ‘Portable Anymap File Format’ (PNM) Toolbox Version 3.0 for MATLAB R13 by Peter J. Acklam obtained from www.mathworks.com/matlabcentral/fileexchange

BIOGRAPHICAL SKETCH

Faizal Iqbal was born on December 6th, 1976 in India. He completed his high school education in Delhi, India. He received his undergraduate degree in the Department of Electrical Engineering at the G B Pant University, Pantnagar in June 1998.

From 1998 to 2001, Faizal was employed by Imr Global (now Part of CGI Inc) in India. During this period, he was responsible for the development and maintenance of business applications running on high end IBM Mainframe and Open systems servers.

He was admitted into the Masters program in the Department of Electrical and Computer Engineering at the Florida State University in fall 2001. His specializes in the fields of Digital IC and ASIC Design.