WAVELET BASED IMAGE COMPRESSION & PERFORMANCE ANALYSIS IN TMS320C67X

BY

KUNAL J SHAH

07MCE020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2009

Wavelet Based Image Compression And Performance Analysis on TMS320C67X

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Kunal J Shah

07MCE020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2009

Certificate

This is to certify that the Major Project entitled " Wavelet Based Image Compression And Performance Analysis on TMES320C67X " submitted by Kunal J Shah (07MCE020), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. S.N. Pradhan
Guide and Professor,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

Digital image processing remains a challenging domain of programming for several reasons. First the issue of digital image processing appeared relatively late in computer history, it had to wait for the arrival of the first graphical operating systems to become a true matter. Secondly, digital image processing requires the most careful optimizations and especially for real time applications. Image processing operations can be roughly divided into three major categories, Image Compression, Image Enhancement and Restoration, and Measurement Extraction. Image compression is familiar to most people. It involves reducing the amount of memory needed to store a digital image. Wavelet is the upcoming technology for the image compression which takes more computations compare to the DCT based image compression. The TMS320C6000 series includes three different type of DSP Chip family TMS320C62X, TMS320C64X and TMS320C67X family. TMS320C64X is the fixed point DSPs while the TMS320C67X is a floating point DSP. Both the TMS320C64XX and TMS320C67XX provides the floating point execution but the TMS320C67X provides more efficient floating point execution.

Acknowledgements

It gives me great pleasure in expressing thanks and profound gratitude to Dr. S. N. Pradhan, PG Coordinator, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout the Major project. I heartily thankful to him for his time to time suggestion and the clarity of the concepts of the topic that helped me a lot during this study.

I would like to give my special thanks to Prof. D.J .Patel, Head of Department , Department of Computer Engineering, Institute of Technology, Nirma University for his continual kind words of encouragement and motivation throughout the Major project. I am also thankful to Dr. Ketan Kotecha, Director, Institute of Technology for his kind support in all respect during my study.

Last but not list I would like to offer my heart-felt gratitude towards my parents for their extreme co-operation and forbearance throughout the project and my friends for their support and understanding they provided me during the project work. I am also very much thankful to all who directly or indirectly helped me.

- Kunal J Shah

07MCE020

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General

Digital image processing remains a challenging domain of programming for several reasons. First the issue of digital image processing appeared relatively late in computer history, it had to wait for the arrival of the first graphical operating systems to become a true matter. Secondly, digital image processing requires the most careful optimizations and especially for real time applications. Comparing image processing and audio processing is a good way to fix ideas. Let us consider the necessary memory bandwidth for examining the pixels of a 320x240, 32 bits bitmap, 30 times a second: 10 Mo/sec. Now with the same quality standard, an audio stereo wave real time processing needs 44100 (samples per second) x 2 (bytes per sample per channel) x 2 (channels) = 176Ko/sec, which is 50 times less. Obviously we will not be able to use the same signal processing techniques in both audio and image. Finally, digital image processing is by definition a two dimensions domain; this somehow complicates things when elaborating digital filters.

Digital image processing is characterized by very high computational demands. Although it can be handled by "standard" computer hardware, such solution is not viable for an embedded system, where dimensions of the computer system, power

consumption or data throughput are of concern. For these reasons, specialized hardware solutions based on a digital signal processor (DSP) or a Field Programmable Gate Array (FPGA) are usually used in embedded system.

Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subfield of digital signal processing, digital image processing has many advantages over analog image processing; it allows a much wider range of algorithms to be applied to the input data, and can avoid problems such as the build-up of noise and signal distortion during processing.

Image processing operations can be roughly divided into three major categories, Image Compression, Image Enhancement and Restoration, and Measurement Extraction. Image compression is familiar to most people. It involves reducing the amount of memory needed to store a digital image.

The heart of any digital signal processing architecture is the Multiply-and Accumulate (MAC) unit. Most signal processing applications utilize a great deal of multiplication: The MAC unit of a DSP accelerates this type of calculation by performing the multiplication of two numbers and then adding the result to all of the previous multiplications in what is called an "accumulator". Another key enabling technology of DSPs is the ability to process several operations at the same time. One way that DSPs can execute four operations at the same time is to use what is known as Very Long Instruction Word (VLIW) architecture. A VLIW is a single instruction that actually represent several operations. DSPs have typically been used to implement many of these applications. Although DSPs are programmable through software, the DSPs' hardware architecture is not flexible. Therefore, DSPs are limited by fixed hardware architecture such as bus performance bottlenecks, a fixed number of MAC blocks, fixed memory, fixed hardware accelerator blocks and fixed data widths. The DSPs' fixed hardware architecture is not suitable for algorithms that want to exploit parallelism in either data or instruction. Compressing an image is significantly different than compressing raw binary data. Of course, general purpose compression programs can be used to compress images, but the result is less

than optimal. 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 a little more bandwidth or storage space. This also means that lossy compression techniques can be used in this area. Lossless compression involves with compressing data which, when decompressed, will be an exact replica of the original data. This is the case when binary data such as executables, documents etc. are compressed. They need to be exactly reproduced when decompressed. On the other hand, images (and music too) need not be reproduced 'exactly'. An approximation of the original image is enough for most purposes, as long as the error between the original and the compressed image is tolerable.

1.2 Motivation

Image processing is a one of the fast developing research area of computer vision. Faster image processing is very essential in current scenario for work automation .This work is useful in developing the vision using the computerized analysis , object detection and classification of the images captured by the sensors for better interpretation and analysis. Generally satellite images are of high resolution 4Kx4K, 16Kx16K and higher resolutions. To support the processing of these images we require special purpose processors like DSPs or ASPs specifically designed for these types of applications. The change from the cine film to digital methods of image exchange and archival is primarily motivated by the ease and flexibility of handling digital image information instead of the film media. While preparing this step and developing standards for digital image communication, one has to make absolutely sure that also the image quality of coronary angiograms and ventriculograms is maintained or improved. Similar requirements exist also in echocardiography. Regarding image quality, the most critical step in going from the analog world (cine film or high definition live video in the catheterization laboratory) to the digital world is the digitization of the signals. For this step, the basic requirement of maintaining image quality is easily translated

into two basic quantitative parameters:

a. the rate of digital image data transfer or data rate (Megabit per second or Mb/s)

b. and the total amount of digital storage required or data capacity (Megabyte or MByte)

As a specific example, the spatial resolution of the cine film is generally assumed to be equivalent to a digital matrix of at least 1000 by 1000 pixels, each with up to 256 gray levels (8 bit or one byte) of contrast information (see Syllabus Unit 1). The following table derives from this principal parameter some examples for requirements on digital image communication and archival in a catheterization laboratory with low to medium volume.

Computer technology, however, provides flexible principles for processing large amounts of information. Among the algorithms available is image data reduction or image compression. The principal approach in data compression is the reduction of the amount of image data (bits) while preserving information (image details). This technology is a key enabling factor in many imaging and multimedia concepts outside of medicine. So one has to ask if cardiology really will have to cope with these enormous and totally uncommon requirements concerning digital data rates and digital data capacity, or if image compression can also be applied without problems in cardiac imaging. At a closer look one observes that ad hoc approaches to image data compression have been applied in most digital imaging systems for the catheterization laboratory all the time. An example is recording the x-ray images with a smaller matrix of just 512 by 512 pixels (instead of the 1024 by 1024 pixel matrix often applied for real-time displays). In order to objectively assess these and other techniques of image data compression, some systematic knowledge of the tradeoffs implied in different modes of image data reduction is mandatory.

1.3 Scope Of Work

As the title suggests the goal of this dissertation focuses on the providing solution for the image compression which is based on the wavelet which is the going beyond the standard jpeg compression. Wavelet is the upcoming technology for the image compression which takes more computations compare to the DCT based image compression. The procedure for the thesis starts with the understanding what is wavelet and DWT(Discrete wavelet Transform). Then the second step will be the selection of the image processing algorithm, which includes the review of the all the algorithms avaliable based on the wavelet transform to be reviewed. Then by selecting some particular algorithm based upon some of the characteristics of the algorithm the implementation of the algorithm takes place. The implementation of the algorithm will be basically on the two platforms like MATLAB and the Code Composer Studio's targetted platform. The selection of the DSP processor to implement the algorithm is also the issue be solved. Basically the TMS320C6000 series includes three different type of DSP Chip family TMS320C62XX, TMS320C64XX and TMS320C67XX family. TMS320C64XX is the fixed point DSPs while the TMS320C67XX is a floating point DSP.

Both the TMS320C64XX and TMS320C67XX provides the floating point execution but the TMS320C67XX provides more efficient floating point execution. As the wavelet tranform will have so much of the calculation the target DSP must have a heavy floating point support. So it will be better to choose the TMS320C67xx architecture to implement the wavelet based image processing algorithms because it demands the support for the floating point calculations.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2, *Literature Survey* , Literature Survey describes the Wavelets, DWT(Discrete Wavelet Transform) and also showing internal details about the flow of the process and the requirements of wavelets for the Image Processing. It also covers the DSP processor background require for the project. It also shows the basics about the Image Processing and Image compression.

Chapter 3, *Image Compression and Algorithms*, This chapter describes the basics of the Image processing and Image compression. It also describes some of the algorithms of the Image Compression. This chapter also descibes the wavelet based image compression techniques and why they are used.

Chapter 4, *Implementation*, This Chapter includes the implementation of the Embedded Image Compression(EZW) algorithm.

Chapter 5, *Experimental Result and Analysis*,

This chapter shows the experimental results obtained using the image compression algorithm implenmentation and further analysis of the results.

Chapter 6, *Conclusion & Future Work*, This chapter describes the future work to be done in the thesis.

Chapter 2

Literature Survey

2.1 Introduction to Image Compression

2.1.1 Image Compression Technique

Compression takes an input X and generates a representation XC that hopefully requires fewer bits. There is a reconstruction algorithm that operates on the compressed representation XC to generate the reconstruction Y. Based on the requirements of reconstruction, data compression schemes can be divided into two broad classes. One is lossless compression, in which Y is identical to X. Examples of lossless methods are Run Length coding, Huffman coding, Lempel/Ziv algorithms, and Arithematic coding. The other is lossy compression, which generally provides much higher compression than lossless compression but allows Y to be different from X.

2.1.2 Loseless Compression Methods

If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. It is generally used for applications that cannot allow any difference between the original and reconstructed data.

Run Length Encoding

Run length encoding, sometimes called recurrence coding, is one of the simplest data compression algorithms. It is effective for data sets that are comprised of long sequences of a single repeated character. For instance, text files with large runs of spaces or tabs may compress well with this algorithm. Old versions of the arc compression program used this method.

RLE finds runs of repeated characters in the input stream and replaces them with a three-byte code. The code consists of a flag character, a count byte, and the repeated characters. For instance, the string "AAAAAABBBBCCCCC" could be more efficiently represented as "*A6*B4*C5". That saves us six bytes. Of course, since it does not make sense to represent runs less than three characters in length with a code, none is used. Thus "AAAAAABBCCCDDDD" might be represented as "*A6BBCCC*D4". The flag byte is called a sentinel byte.

Huffman Coding

Huffman coding, developed by D.A. Huffman, is a classical data compression technique. It has been used in various compression applications, including image compression. It uses the statistical property of characters in the source stream and then produces respective codes for these characters. These codes are of variable code length using an integral number of bits. The codes for characters having a higher frequency of occurrence are shorter than those codes for characters having lower frequency. This simple idea causes a reduction in the average code length, and thus the overall size of compressed data is smaller than the original. Huffman coding is based on building a binary tree that holds all characters in the source at its leaf nodes, and with their corresponding characters' probabilities at the side. The tree is built by going through the following steps:

a. Each of the characters is initially laid out as leaf node; each leaf will eventually be connected to the tree. The characters are ranked according to their weights,

which represent the frequencies of their occurrences in the source.

b. Two nodes with the lowest weights are combined to form a new node, which is a parent node of these two nodes. This parent node is then considered as a representative of the two nodes with a weight equal to the sum of the weights of two nodes. Moreover, one child, the left, is assigned a "0" and the other, the right child, is assigned a "1".

c. Nodes are then successively combined as above until a binary tree containing all of nodes is created.

d. The code representing a given character can be determined by going from the root of the tree to the leaf node representing the alphabet. The accumulation of "0" and "1" symbols is the code of that character.

By using this procedure, the characters are naturally assigned codes that reflect the frequency distribution. Highly frequent characters will be given short codes, and infrequent characters will have long codes. Therefore, the average code length will be reduced. If the count of characters is very biased to some particular characters, the reduction will be very significant.

Lempel-Ziv-Welch(LZW) Encoding

This original approach is given by J. Ziv and A. Lempel in 1977. T. Welch's refinements to the algorithm were published in 1984. LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds.

There are three best-known applications of LZW:

- UNIX Compress (file compression)
- GIF (image compression)

Arithmetic Coding

Arithmetic coding is also a kind of statistical coding algorithm similar to Huffman coding. However, it uses a different approach to utilize symbol probabilities, and performs better than Huffman coding. In Huffman coding, optimal codeword length is obtained when the symbol probabilities are of the form $\frac{1}{2}^x$, where x is an integer. This is because Huffman coding assigns code with an integral number of bits. This form of symbol probabilities is rare in practice. Arithmetic coding is a statistical coding method that solves this problem. The code form is not restricted to an integral number of bits. It can assign a code as a fraction of a bit. Therefore, when the symbol probabilities are more arbitrary, arithmetic coding has a better compression ratio than Huffman coding. In brief, this is can be considered as grouping input symbols and coding them into one long code. Therefore, different symbols can share a bit from the long code.

Although arithmetic coding is more powerful than Huffman coding in compression ratio, arithmetic coding requires more computational power and memory. Huffman coding is more attractive than arithmetic coding when simplicity is the major concern.

2.1.3 Lossy Compression Methods

Lossy compression techniques involve some loss of information, and data cannot be recovered or reconstructed exactly. In some applications, exact reconstruction is not necessary. For example, it is acceptable that a reconstructed video signal is different from the original as long as the differences do not result in annoying artifacts. However, we can generally obtain higher compression ratios than is possible with lossless compression.

Vector Quantization

Vector Quantization (VQ) is a lossy compression method. It uses a codebook containing pixel patterns with corresponding indexes on each of them. The main idea of VQ is to represent arrays of pixels by an index in the codebook. In this way, compression is achieved because the size of the index is usually a small fraction of that of the block of pixels. The main advantages of VQ are the simplicity of its idea and the possible efficient implementation of the decoder. Moreover, VQ is theoretically an efficient method for image compression, and superior performance will be gained for large vectors. However, in order to use large vectors, VQ becomes complex and requires many computational resources (e.g. memory, computations per pixel) in order to efficiently construct and search a codebook. More research on reducing this complexity has to be done in order to make VQ a practical image compression method with superior quality.

Predictive Coding

Predictive coding has been used extensively in image compression. Predictive image coding algorithms are used primarily to exploit the correlation between adjacent pixels. They predict the value of a given pixel based on the values of the surrounding pixels. Due to the correlation property among adjacent pixels in image, the use of a predictor can reduce the amount of information bits to represent image. This type of lossy image compression technique is not as competitive as transform coding techniques used in modern lossy image compression, because predictive techniques have inferior compression ratios and worse reconstructed image quality than those of transform coding.

Fractal Compression

The application of fractals in image compression started with M.F. Barnsley and A. Jacquin. Fractal image compression is a process to find a small set of mathematical

equations that can describe the image. By sending the parameters of these equations to the decoder, we can reconstruct the original image.

In general, the theory of fractal compression is based on the contraction mapping theorem in the mathematics of metric spaces. The Partitioned Iterated Function System (PIFS), which is essentially a set of contraction mappings, is formed by analysing the image. Those mappings can exploit the redundancy that is commonly present in most images. This redundancy is related to the similarity of an image with itself, that is, part A of a certain image is similar to another part B of the image, by doing an arbitrary number of contractive transformations that can bring A and B together. These contractive transformations are actually common geometrical operations such as rotation, scaling, skewing and shifting. By applying the resulting PIFS on an initially blank image iteratively, we can completely regenerate the original image at the decoder. Since the PIFS often consists of a small number of parameters, a huge compression ratio (e.g. 500 to 1000 times) can be achieved by representing the original image using these parameters. However, fractal image compression has its disadvantages. Because fractal image compression usually involves a large amount of matching and geometric operations, it is time consuming. The coding process is so asymmetrical that encoding of an image takes much longer time than decoding.

Transform Based Image Compression. The basic encoding method for transform based compression works as follows:

a. Image transform

   Divide the source image into blocks and apply the transformations to the blocks.

b. Parameter quantization

   The data generated by the transformation are quantized to reduce the amount of information. This step represents the information within the new domain by reducing the amount of data. Quantization is in most cases not a reversible operation because of its lossy property.

c. Encoding

Encode the results of the quantization. This last step can be error free by using Run Length encoding or Huffman coding. It can also be lossy if it optimizes the representation of the information to further reduce the bit rate.

Transform based compression is one of the most useful applications. Combined with other compression techniques, this technique allows the efficient transmission, storage, and display of images that otherwise would be impractical.

DCT-based Transform Coding

The Discrete Cosine Transform (DCT) was first applied to image compression in the work by Ahmed, Natarajan, and Rao. It is a popular transform used by the JPEG (Joint Photographic Experts Group) image compression standard for lossy compression of images. Since it is used so frequently, DCT is often referred to in the literature as JPEG-DCT, DCT used in JPEG.

JPEG-DCT is a transform coding method comprising four steps. The source image is first partitioned into sub-blocks of size 8x8 pixels in dimension. Then each block is transformed from spatial domain to frequency domain using a 2-D DCT basis function. The resulting frequency coefficients are quantized and finally output to a lossless entropy coder. DCT is an efficient image compression method since it can decorrelate pixels in the image (since the cosine basis is orthogonal) and compact most image energy to a few transformed coefficients. Moreover, DCT coefficients can be lossily quantized according to some human visual characteristics. Therefore, the JPEG image file format is very efficient. This makes it very popular, especially in the World Wide Web. However, JPEG may be replaced by wavelet-based image compression algorithms, which have better compression performance.

2.2 Wavelet Transform

Wavelets are functions defined over a finite interval. The basic idea of the wavelet transform is to represent an arbitrary function (x) as a linear combination of a set of such wavelets or basis functions. These basis functions are obtained from a single prototype wavelet called the mother wavelet by dilations (scaling) and translations (shifts).

The purpose of wavelet transform is to change the data from time-space domain to time-frequency domain which makes better compression results. The simplest form of wavelets, the Haar wavelet function **??** is defined as:

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \leq x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

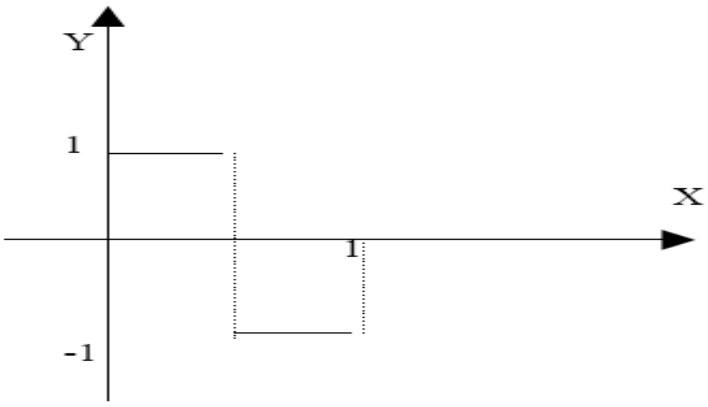

Figure 2.1: Haar Wavelet

The following is a simple example to show how to perform Haar wavelet transform on four sample numbers. Assume we have four numbers

$x(0) = 1.2$, $x(1) = 1.0$, $x(2) = -1.0$, $x(3) = -1.2$

Let us perform Haar wavelet transform on these four numbers.

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 2.2 \\ 0.2 \\ -2.2 \\ 0.2 \end{bmatrix}$$

Notice we can always do inverse transform from x to y:

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix}$$

If 0.2 is below our quantization threshold, it will be replaced by 0. Then, reconstructed x will be [1.1, 1.1, -1.1, -1.1]. After first transform, we keep y(1) and y(3) at the finest level and iterate the transform on y(0) and y(2) again. $z(0) = y(0)+y(2) = 0$ and $z(2) = y(0) - y(2) = 4.4$.

Those four numbers become [0, 0.2, 4.4, 0.2]. After quantization, they could be [0, 0, 4, 0], which are much easier to be compressed.

2.2.1 Why to use Wavelet Based Compression

As discussed earlier, for image compression, loss of some information is acceptable. Among all of the above lossy compression methods, vector quantization requires many computational resources for large vectors; fractal compression is time consuming for coding; predictive coding has inferior compression ratio and worse reconstructed image quality than those of transform based coding. So, transform based compression methods are generally best for image compression. For transform based compression, JPEG compression schemes based on DCT (Discrete Cosine Transform) have some advantages such as simplicity, satisfactory performance, and availability of special purpose hardware for implementation. However, because the input image is blocked, correlation across the block boundaries cannot be eliminated.

Over the past ten years, the wavelet transform has been widely used in signal processing research, particularly, in image compression. In many applications, wavelet-based schemes achieve better performance than other coding schemes like the one based on DCT. Since there is no need to block the input image and its basis functions have variable length, wavelet based coding schemes can avoid blocking artifacts. Wavelet based coding also facilitates progressive transmission of images.

2.2.2 Comparison of Wavelet Properties

The following table shows the property comparison of three kinds of wavelets.

Property	Haar	Daubechie	Biorthogonal Spline
Explicit Function	Yes	No	Yes
Orthogonal	Yes	Yes	No
Symmetric	Yes	No	Yes
Continuous	No	Yes	Yes
Compacted support	Yes	Yes	Yes
Maximum regularity for order L	No	No	Yes
Shortest scaling function for order L	Yes	No	Yes

Table I: Property Comparison of Three Kinds of Wavelets

Haar and Daubechies wavelets have orthogonality, which has some nice features.

a. The scaling function and wavelet function are the same for both forward and inverse transform.

b. The correlations in the signal between different subspaces are removed.

Among the three kinds of wavelets, the Haar wavelet transform is the simplest one to implement, and it is the fastest. The major disadvantage of the Haar wavelet is its discontinuity, which makes it difficult to simulate a continuous signal. Daubechie found the first continuous orthogonal compact support wavelet. Note that this family

of wavelets is not symmetric. The advantage of symmetry is that the corresponding wavelet transform can be implemented using mirror boundary conditions that reduces boundary artifacts. That is why we introduce the biorthogonal spline wavelet. For the biorthogonal spline wavelet, the scaling function is a B-spline. The B-spline of degree N is the shortest possible scaling function of order N-1 and B-splines are the smoothest scaling functions for a filter of a given length. Because splines are piecewise polynomial, they are easy to manipulate. For example, it is simple to get spline derivatives and integrals.

2.3 TMS320C6000 DSP Family

2.3.1 Introduction

The TMS320C6000 platform of digital signal processors (DSPs) is part of the TMS320 family of DSPs. The TMS320C62x ('C62x) devices are fixed-point DSPs in the TMS320C6000 platform. The TMS320C67x ('C67x) devices are floating-point DSPs in the TMS320C6000 platform. The TMS320C62x and TMS320C67x are code compatible and both feature the VelociTI architecture.

The VelociTI architecture is a high-performance, advanced, very-long-instruction-word (VLIW) architecture developed by Texas Instruments, making these DSPs excellent choices for multi channel and multifunction applications. VelociTI, together with the development tool set and evaluation tools, provides faster development time and higher performance for embedded DSP applications through increased instruction-level parallelism. With performance of up to 2000 million instructions per second (MIPS) at 250 MHz and a complete set of development tools, the TMS320C6000 DSPs offer cost-effective solutions to high-performance DSP programming challenges. The TMS320C6000 development tools include a new C compiler, an assembly optimizer that simplifies programming and scheduling, and a Windows debugger interface.

The TMS320DSP family consists of fixed-point, floating-point, and multiproces-

sor digital signal processors (DSPs). TMS320DSPs have an architecture designed specifically for real-time signal processing.

With a performance of up to 6000 million instructions per second (MIPS) and an efficient C compiler, the TMS320C6000 DSPs [4] give system architects unlimited possibilities to differentiate their products. High performance, ease of use, and affordable pricing make the C6000 generation the ideal solution for multifunction applications, such as, 3-D transformations, Image compression/transmission, Image enhancement, Pattern recognition, Robot vision, etc. The newest member of the C6000 family, the C64x, brings the highest level of performance for processing data in this era of data convergence. At clock rates of 600 MHz and greater, the C64x can process information at a rate of 4800-6400 MIPS. In addition to clock rate, more work can be done each cycle with the VelociTI architecture. These extensions include new instructions to accelerate performance in key applications and extend the parallelism of the architecture. Increased clock rate and increased CPU throughput are only part of the solution. Processing data at these extremely high rates increases the need for I/O bandwidth.

Three flexible Multi-channel Buffered Serial Ports can each supply 100Mbits/sec each of additional throughput. The internal DMA engine can provide over 2Gbytes/sec of I/O bandwidth with 64 independent channels. The C64x goes beyond a core and peripheral set to bring the maximum level of performance for processing digital data quickly. The tight coupling of the CPU architecture and the compiler help to maximize processor throughput. The RISC like instruction set and extensive use of pipelining allow many instructions to be scheduled and executed in parallel. The key extensions made to the 'C62x architecture that allow the 'C64x to perform more work each clock cycle include wider data paths, a larger register file, greater orthogonality and new instructions that support packed data processing.

The TMS320C6000 DSPs give system architects unlimited possibilities to differentiate their products. High performance, ease of use, and affordable pricing make the TMS320C6000 platform the ideal solution for multichannel, multifunction appli-

cations, such as:

- Pooled modems
- Wireless local loop base stations
- Beam-forming base stations
- Remote access servers (RAS)
- Digital subscriber loop (DSL) systems
- Cable modems
- Multichannel telephony systems
- Virtual reality 3-D graphics
- Speech recognition
- Audio
- Radar
- Atmospheric modeling
- Finite element analysis
- Imaging (examples: fingerprint recognition, ultrasound, and MRI)

The TMS320C6000 platform is also an ideal solution for exciting new applications; for example:

- Personalized home security with face and hand/fingerprint recognition
- Advanced cruise control with global positioning systems (GPS) navigation and accident avoidance
- Remote medical diagnostics

2.3.2 TMS320C67x Series DSP

The C6000 devices execute up to eight 32-bit instructions per cycle. The C67x CPU consists of 32 general-purpose 32-bit registers and eight functional units. These eight functional units contain:

- Two Multipliers
- Six ALUs

The C6000 generation has a complete set of optimized development tools, including an efficient C compiler, an assembly optimizer for simplified assembly-language programming and scheduling, and a Windows based debugger interface for visibility into source code execution characteristics. A hardware emulation board, compatible with the TI XDS510 and XDS56 emulator interface, is also available. This tool complies with IEEE Standard 1149.1?1990, IEEE Standard Test Access Port and Boundary-Scan Architecture. Features of the C6000 devices include:

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units
- Executes up to eight instructions per cycle for up to ten times the performance of typical DSPs
- Allows designers to develop highly effective RISC-like code for fast development time
- Instruction packing
- Gives code size equivalence for eight instructions executed serially or in parallel
- Reduces code size, program fetches, and power consumption
- Conditional execution of all instructions
- Reduces costly branching

- Increases parallelism for higher sustained performance
- Efficient code execution on independent functional units
- Industry's most efficient C compiler on DSP benchmark suite
- Industry's first assembly optimizer for fast development and improved parallelization
- 8/16/32-bit data support, providing efficient memory support for a variety of applications
- 40-bit arithmetic options add extra precision for vocoders and other computationally intensive applications
- Saturation and normalization provide support for key arithmetic operations
- Field manipulation and instruction extract, set, clear, and bit counting support common operation found in control and data manipulation applications.

The C67x devices include these additional features:

- Hardware support for single-precision (32-bit) and double-precision (64-bit) IEEE floating-point operations.
- 32 32-bit integer multiply with 32-bit or 64-bit result.

In addition to the features of the C67x device, the C67x+ device is enhanced for code size improvement and floating-point performance. These additional features include:

- Execute packets can span fetch packets.
- Register file size is increased to 64 registers (32 in each datapath).
- Floating-point addition and subtraction capability in the .S unit.

- Mixed-precision multiply instructions.
- 32-KByte instruction cache that supports execution from both on-chip RAM and ROM as well as from external memory through a VBUSP-based external memory interface (EMIF).
- Unified memory controller features support for flat on-chip data RAM and ROM organizations for zero wait-state accesses from both load store unit of the CPU. The memory controller supports different banking organizations for RAM and ROM arrays. The memory controller also supports VBUSP interfaces (two master and one slave) for transfer of data from the system peripherals to and from the CPU and internal memory. A VBUSP based DMA controller can interface to the CPU for programmable bulk transfers through the VBUSP slave port.

2.3.3 TMS320C67x DSP Architecture

Figure 2.2 is the block diagram for the C67x DSP. The C6000 devices come with program memory, which, on some devices, can be used as a program cache. The devices also have varying sizes of data memory. Peripherals such as a direct memory access (DMA) controller, power-down logic, and external memory interface (EMIF) usually come with the CPU, while peripherals such as serial ports and host ports are on only certain devices.

Central Processing Unit(CPU)

The C67x CPU, in Figure 2.2, is common to all the C62x/C64x/C67x [1] devices. The CPU contains:

- Program fetch unit
- Instruction dispatch unit

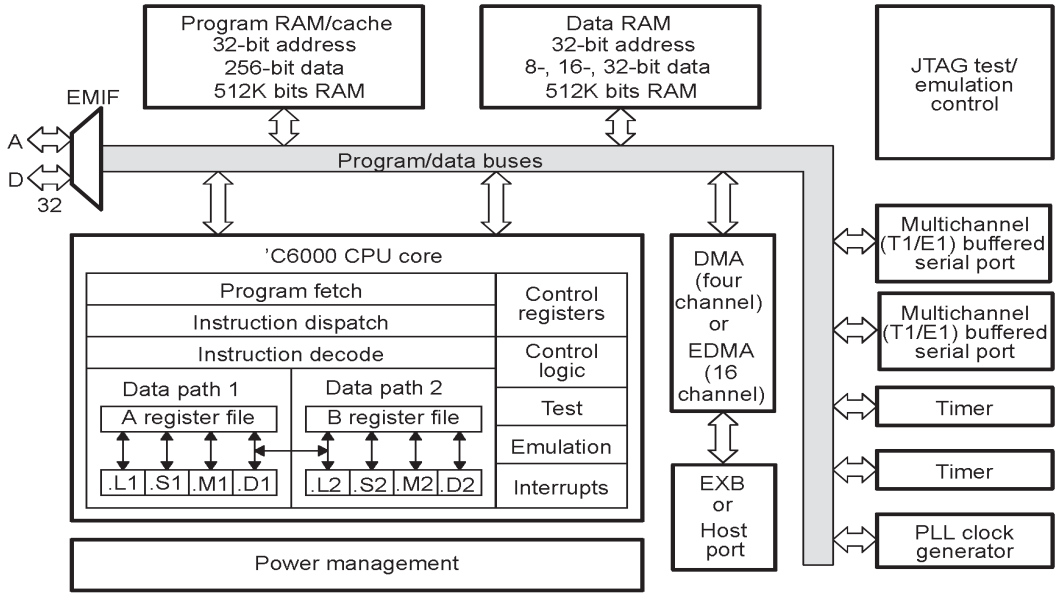

Figure 2.2: TMS320C67x DSP Block Diagram

- Instruction decode unit
- Two data paths, each with four functional units
- 32 32-bit registers
- Control registers
- Control logic
- Test, emulation, and interrupt logic

The program fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B), each of which contains four functional units (.L, .S, .M, and .D) and 16 32-bit general-purpose registers. A control register file provides the means to configure and control various processor operations.

Internal Memory

The C67x DSP has a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces. When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF). The C67x DSP has two 32-bit internal ports to access internal data memory. The C67x [2] DSP has a single internal port to access internal program memory, with an instruction-fetch width of 256 bits.

Memory and Peripheral Options

A variety of memory and peripheral options are available for the C6000 platform:

- Large on-chip RAM, up to 7M bits
- Program cache
- 2-level caches
- 32-bit external memory interface supports SDRAM, SBSRAM, SRAM, and other asynchronous memories for a broad range of external memory requirements and maximum system performance.
- DMA Controller (C6701 DSP only) transfers data between address ranges in the memory map without intervention by the CPU. The DMA controller has four programmable channels and a fifth auxiliary channel.
- EDMA Controller performs the same functions as the DMA controller. The EDMA has 16 programmable channels, as well as a RAM space to hold multiple configurations for future transfers.
- HPI is a parallel port through which a host processor can directly access the CPU's memory space. The host device has ease of access because it is the master of the interface. The host and the CPU can exchange information via internal

or external memory. In addition, the host has direct access to memory-mapped peripherals.

- Expansion bus is a replacement for the HPI, as well as an expansion of the EMIF. The expansion provides two distinct areas of functionality (host port and I/O port) which can co-exist in a system. The host port of the expansion bus can operate in either asynchronous slave mode, similar to the HPI, or in synchronous master/slave mode. This allows the device to interface to a variety of host bus protocols. Synchronous FIFOs and asynchronous peripheral I/O devices may interface to the expansion bus.

- McBSP (multichannel buffered serial port) is based on the standard serial port interface found on the TMS320C2000 and TMS320C5000 devices. In addition, the port can buffer serial samples in memory automatically with the aid of the DMA/EDNA controller. It also has multichannel capability compatible with the T1, E1, SCSA, and MVIP networking standards.

- Timers in the C6000 devices are two 32-bit general-purpose timers used for these functions:

  - Time events
  - Count events
  - Generate pulses
  - Interrupt the CPU
  - Send synchronization events to the DMA/EDMA controller.

- Power-down logic allows reduced clocking to reduce power consumption. Most of the operating power of CMOS logic dissipates during circuit switching from one logic state to another. By preventing some or all of the chip's logic from switching, you can realize significant power savings without losing any data or operational context.

Memory System

The memory system of the TMS320C67x implements a modified Harvard architecture, providing separate address spaces for instruction and data memory. The TMS320C67x fetches instructions using a 32-bit address bus and 256-bit data bus. Each data path accesses data using a 32-bit address bus and a 64-bit data bus. Together, these data buses can perform two 64-bit loads, two 32-bit stores, or a 64-bit load and a 32-bit store to or from on-chip memory per instruction cycle. The TMS320C67x allows up to two data move instructions to be executed in parallel with other instructions. The TMS320C6701 contains 64 Kbytes of program RAM and 64 Kbytes of data RAM. The program memory can be configured to act as an instruction cache. The TMS320C6711 and TMS320C6712 use the same two-level on-chip cache memory architecture as the fixed-point TMS320C6211. The TMS320C6711 and TMS320C6712 each contain two 4 Kbyte level-one caches, one for data and one for instructions. The level-one caches are fed by a unified 64 Kbyte level-two memory. The level-two memory can be configured as SRAM, as a cache, or as a partitioned combination of the two. The TMS320C6713 on-chip memory configuration is similar to that of the TMS320C6711 and TMS320C6712, except that it contains a larger 256 Kbyte level-two memory.

Addressing

The TMS320C67x supports register-direct and register-indirect addressing modes and immediate data. In register-indirect addressing mode, the address register modification options include pre-increment/decrement by a short (5bit) immediate or by the contents of any general-purpose register, and post-increment/decrement by a short immediate or by the contents of any general-purpose register. The TMS320C67x supports modulo addressing. Up to eight registers (four from each register file) can be configured to operate under modulo addressing. The TMS320C67x does not support bit-reversed addressing.

Pipeline

The TMS320C67x pipeline consists of 16 stages. The pipeline is non-interlocked and is significantly deeper than those of other commercially available DSP processors. Instructions are always fetched eight at a time via the 256-bit instruction bus. This group of eight instructions is called a "fetch packet." However, the TMS320C67x cannot always execute eight instructions in parallel. The group of instructions to be executed in parallel is called an "execution packet." Because the TMS320C67x supports variable-length execution packets (and thus can execute from one to eight instructions in parallel), a single fetch packet may contain several execution packets. The processor does not check execution packets for resource contention. Consequently, hand-written assembly code may introduce resource conflicts that produce unwanted behavior.

All branches on the TMS320C67x are delayed branches with five delay slots. Most fixed-point instructions on the TMS320C67x have a latency of one cycle. The branch, fixed-point multiply, and load instructions produce results only after several cycles. Latencies for floating point operations range from one to ten cycles.

Instruction Set

The TMS320C62x uses an opcode- operand assembly language format where each instruction has an opcode field for the operation and an operand field for one to four operands. In addition, three optional fields can be used to indicate parallel execution, conditional execution, and the targeted execution unit. If the target execution unit field is omitted from the instruction, the assembler attempts to select an appropriate execution unit. All instructions on the TMS320C67x can be executed conditionally. Five designated general-purpose registers can be used as condition registers. The TMS320C67x does not support hardware looping, so all loops must be implemented in software. However, the parallel architecture of the processor allows the implementation of software loops with virtually no overhead. Due to its simple,

RISC-like instructions, 32-bit instruction width, and uniform register sets, the instruction set of the TMS320C67x is extremely regular and straightforward. Because the TMS320C67x is a highly parallel architecture, obtaining maximum performance often requires the programmer to schedule instructions carefully. This can be a challenge because the TMS320C67x has a complex architecture and long, variable instruction latencies. Texas Instruments' assembly optimizer tools and C compiler simplify code development by automating the scheduling and parallelization processes, but these tools do not always result in optimal code.

Benchmark Performance

The BDTI BenchmarksTM are a set of DSP software functions that BDTI has independently designed to provide an objective basis for comparing processor performance characteristics such as speed and memory use for DSP applications. The BDTI Benchmark functions are implemented in optimized assembly language to allow a realistic assessment of processors' signal processing performance. The resulting software is then verified for functional correctness, opti-mality, and adherence to the BDTI Benchmark specifications. Benchmark performance results are obtained either through manual analysis and careful, detailed simulation, or by measurement on sample devices. BDTI's reports such as Buyer's Guide to DSP Processors and the Inside series of reports include extensive BDTI Benchmark results used to evaluate the DSP performance of a set of processors. For each benchmark, BDTI typically reports cycle counts, execution time, a cost-performance metric, an energy-efficiency metric, and memory usage. In this section, we present sample execution time, cost-performance, energy consumption, and memory usage results taken from BDTI's library of benchmark results for the TMS320C67x and two other floating-point processors: the Analog Devices ADSP-2116x and the Renesas (formerly Hitachi) SH775x, which is based on the SH-4 core.

Execution Time:

Execution time results in this report were obtained assuming instructions and data are preloaded in caches where applicable. Processor speeds are for the fastest available chips as of mid-2003. Figure 2.3 shows execution time results on BDTI's 256-point FFT benchmark for the fastest member of each processor family. The 225 MHz TMS320C6713 is over 30% faster on this benchmark than either the 100 MHz ADSP-21161N or the 240 MHz SH7750R. The TMS320C6713 achieves this speed through a combination of a high clock rate and a high degree of parallelism.

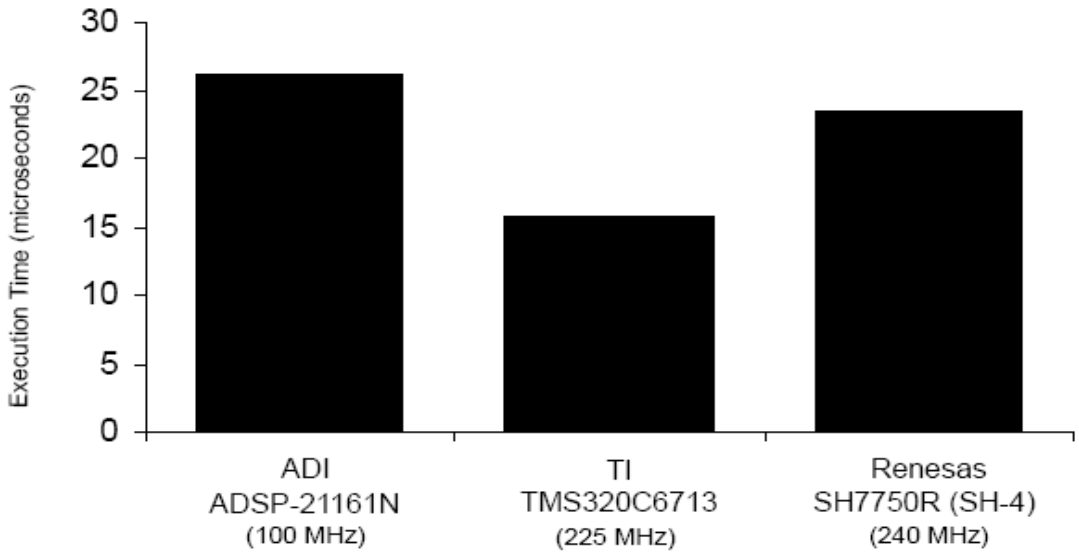

Figure 2.3: 256-point FFT Benchmark Execution times (lower is better)

The TMS320C6713 could be even faster if it supported floating-point SIMD instructions like those supported by the ADSP-21161N and the SH7750R. For example, the ADSP-21161N can compute both the sum and the difference of two operands with a single instruction. The TMS320C6713 is also held back by the long latencies of its floating-point arithmetic instructions (for example, four cycles for single-precision additions and multiplications). As mentioned above, the ADSP-21161N has instructions that accelerate the FFT butterfly. However, the ADSP-21161N has a relatively low

clock speed, so it cannot keep up with the TMS320C6713. The SH7750R is also quite fast at processing the FFT butterfly. However, the SH7750R spends many cycles loading coefficients between FFT butterflies. Consequently, the SH7750R is much slower than the TMS320C6713 on this benchmark.

Cost-Performance

Figure 2.4 shows cost-performance results for the most cost-effective member of each processor family. To create the cost-performance metric, the FFT execution time is multiplied by the cost of the processor in 10,000-unit quantities as of mid-2003. Based on this analysis, the 150 MHz TMS320C6712C is the most cost-effective processor considered here. It is over 35% more cost-effective than either the 100 MHz ADSP-21161N or the 200 MHz SH7750R. It should be noted that included on-chip memory and peripherals can be a significant factor in overall cost. These factors are not considered in the cost-performance metric used here.

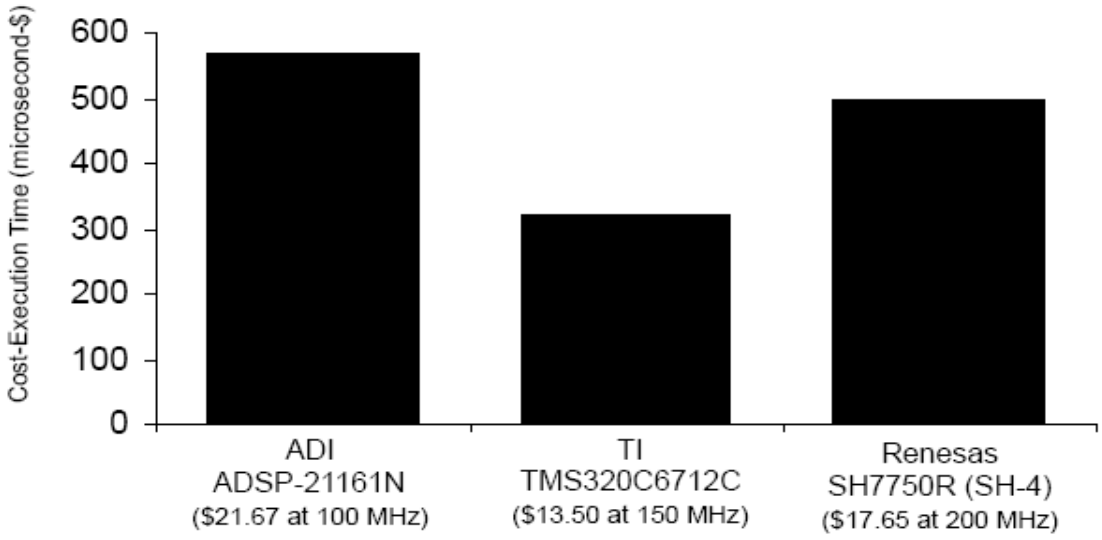

Figure 2.4: Cost Execution Time Product for 256-point FFT(lower is better)

Energy Efficiency

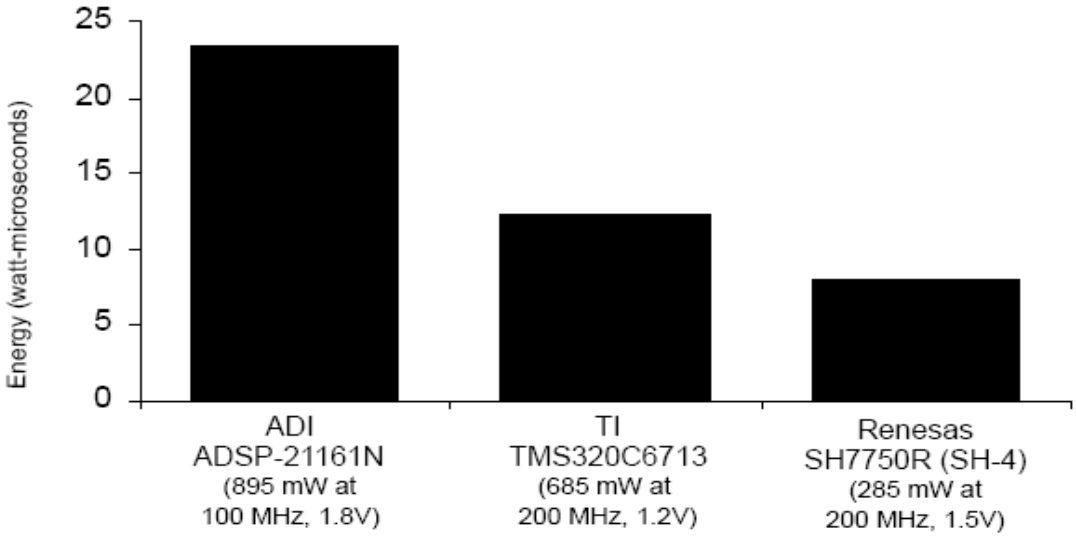

Figure 2.5: Energy Consumption for 256-point FFT(lower is better)

Figure 2.5 shows energy consumption results for the most energy-efficient member of each processor family. To estimate the energy consumption metric, the FFT execution time is multiplied by the typical power consumption of the processor. Figure 2.5 lists the typical power consumption for each processor below the processor name. The 200 MHz TMS320C6713 is about 30% faster than the 100 MHz ADSP-21161N, and it consumes about 25% less power than the ADSP-21161N. Consequently, it consumes about half as much energy as the ADSP-21161N. The 200 MHz TMS320C6713 is also about 30% faster than the 200 MHz SH7750R, but it consumes over two times more power than the SH7750R. Hence, the TMS320C6713 consumes about 50% more energy than the SH7750R.

Memory Use

Execution speed is often the primary metric used to compare processors. However, a processor's memory usage is also important. For example, the memory requirements of an application can have a significant impact on overall system cost. In addition, processors may experience significant performance degradation when instructions and data do not fit in on-chip memory. Because of these and other factors, memory efficiency is an important metric in processor selection. For each of the BDTI Benchmarks, BDTI measures each processor's program, constant data, non-constant data, and total memory use.

Chapter 3

Image Compression Algorithms

As our use of and reliance on computers continues to grow, so too does our need for efficient ways of storing large amounts of data. For example, someone with a web page or online catalog - that uses dozens or perhaps hundreds of images - will more than likely need to use some form of image compression to store those images. This is because the amount of space required to hold unadulterated images can be prohibitively large in terms of cost. Fortunately, there are several methods of image compression available today. These fall into two general categories: lossless and lossy image compression.

3.1 JPEG Image Compression

The JPEG process is a widely used form of lossy image compression that centers around the Discrete Cosine Transform. The DCT works by separating images into parts of differing frequencies. During a step called quantization, where part of compression actually occurs, the less important frequencies are discarded, hence the use of the term "lossy." Then, only the most important frequencies that remain are used retrieve the image in the decompression process. As a result, reconstructed images contain some distortion; but as we shall soon see, these levels of distortion can be adjusted during the compression stage. The JPEG method is used for both color and

black-and-white images [3].

3.1.1 Overview

This is the general scenario how the JPEG Compression works.

a. The image is broken into 8x8 blocks of pixels.

b. Working from left to right, top to bottom, the DCT is applied to each block.

c. Each block is compressed through quantization.

d. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space.

e. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).

3.1.2 The DCT Equation

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos[\frac{(2x+1)i\pi}{2N}] \cos[\frac{(2y+1)i\pi}{2N}] \quad (3.1)$$

$$C(u) = \frac{1}{\sqrt{2}} if u = 0; 1 if u > 0; \quad (3.2)$$

p(x,y) is the x,y^th element of the image represented by the matrix p. N is the size of the block that the DCT is done on. The equation calculates one entry (i,j^th) of the transformed image from the pixel values of the original image matrix. For the standard 8X8 block that JPEG compression uses, N equals 8 and x and y range from 0 to 7. Therefore D(i,j) would be as in Equation 3.3.

$$D(i,j) = \frac{1}{4} C(i)C(j) \sum_{x=0}^{7} \sum_{y=0}^{7} p(x,y) \cos[\frac{(2x+1)i\pi}{16}] \cos[\frac{(2y+1)i\pi}{16}] \quad (3.3)$$

Because the DCT uses cosine functions, the resulting matrix depends on the horizontal, diagonal, and vertical frequencies. Therefore an image black with a lot of change in frequency has a very random looking resulting matrix, while an image matrix of just one color, has a resulting matrix of a large value for the first element and zeroes for the other elements.

3.1.3 The DCT Matrix

To get the matrix from Equation 3.1 we will use the following equation,

$$T_{i,j} = \frac{1}{\sqrt{N}} \;\; if \;\; i = 0; \;\; \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N} \;\; if \;\; i > 0; \tag{3.4}$$

For an 8x8 block it results in this matrix:

$$\mathrm{T} = \begin{matrix} .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 \\ .4904 & .4157 & .2778 & .0975 & -.0975 & -.2778 & -.4157 & -.4904 \\ .4619 & .1913 & -.1913 & -.4619 & -.4619 & -.1913 & .1913 & .4619 \\ .4157 & -.0975 & -.4904 & -.2778 & .2778 & .4904 & .0975 & -.4157 \\ .3536 & -.3536 & -.3536 & .3536 & .3536 & -.3536 & -.3536 & .3536 \\ .2778 & -.4904 & .0975 & .4157 & -.4157 & -.0975 & .4904 & -.2778 \\ .1913 & -.4619 & .4619 & -.1913 & -.1913 & .4619 & -.4619 & .1913 \\ .0975 & -.2778 & .4157 & -.4904 & .4904 & -.4157 & .2778 & -.0975 \end{matrix}$$

The first row ($i = 1$) of the matrix has all the entries equal to $\frac{1}{\sqrt{8}}$ as expected from Equation 3.4. The columns of T form an orthonormal set, so T is an orthogonal matrix. When doing the inverse DCT the orthogonality of T is important, as the inverse of T is 7 which is easy to calculate.

3.1.4 DCT on a 8X8 Block

Before we begin, it should be noted that the pixel values of a black-and-white image range from 0 to 255 in steps of 1, where pure black is represented by 0, and pure white by 255. Thus it can be seen how a photo, illustration, etc. can be accurately represented by these 256 shades of gray.

Since an image comprises hundreds or even thousands of 8x8 blocks of pixels, the following description of what happens to one 8x8 block is a microcosm of the JPEG process; what is done to one block of image pixels is done to all of them, in the order earlier specified. Now, let's start with a block of image-pixel values.

Original Image Block:

154	123	123	123	123	123	123	126
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

Because the DCT is designed to work on pixel values ranging from -128 to 127, the original block is "leveled off" by subtracting 128 from each entry. This results in the following matrix.

M =

26	-5	-5	-5	-5	-5	-5	8
64	52	8	26	26	26	8	-18
126	70	26	26	52	26	-5	-5
111	52	8	52	52	38	-5	-5
52	26	8	39	38	21	8	8
0	8	-5	8	26	52	70	26
-5	23	-18	21	8	8	52	38
-18	8	-5	-5	-5	8	26	8

Now the DCT (Discrete Cosine Transform) is performed using following equation.

$$D = TMT' \tag{3.5}$$

In Equation 3.5 matrix M is first multiplied on the left by the DCT matrix T from the previous section; this transforms the rows. The columns are then transformed by multiplying on the right by the transpose of the DCT matrix. This yields the following matrix.

D =

162.3	40.6	20.0	72.3	30.3	12.5	-19.7	-11.5
30.5	108.4	10.5	32.3	27.7	-15.5	18.4	-2.0
-94.1	-60.1	12.3	-43.4	-31.3	6.1	-3.3	7.1
30.5	108.4	10.5	32.3	27.7	-15.5	18.4	-2.0
-94.1	-60.1	12.3	-43.4	-31.3	6.1	-3.3	7.1
30.5	108.4	10.5	32.3	27.7	-15.5	18.4	-2.0
-94.1	-60.1	12.3	-43.4	-31.3	6.1	-3.3	7.1
30.5	108.4	10.5	32.3	27.7	-15.5	18.4	-2.0

This block matrix now consists of 64 DCT coefficients, Cy, where i and j range from 0 to 7. The top-left coefficient, C00, correlates to the low frequencies of the original image block. As we move away from C00 in all directions, the DCT coefficients correlate to higher and higher frequencies of the image block, where C77 corresponds

to the highest frequency. It is important to note that the human eye is most sensitive to low frequencies, and results from the quantization step will reflect this fact.

3.1.5 Quantization

The 8x8 block of DCT coefficients is now ready for compression by quantization. A remarkable and highly useful feature of the JPEG process is that in this step, varying levels of image compression and quality are obtainable through selection of specific quantization matrices. This enables the user to decide on quality levels ranging from 1 to 100, where 1 gives the poorest image quality and highest compression, while 100 gives the best quality and lowest compression. As a result, the quality/compression ratio can be tailored to suit different needs. Subjective experiments involving the human visual system have resulted in the JPEG standard quantization matrix. With a quality level of 50, this matrix renders both high compression and excellent decompressed image quality.

$Q_{50} =$

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

If, however, another level of quality and compression is desired, scalar multiples of the JPEG standard quantization matrix may be used. For a quality level greater than 50 (less compression, higher image quality), the standard quantization matrix is multiplied by (100-quality level)/50. For a quality level less than 50 (more compression, lower image quality), the standard quantization matrix is multiplied by 50/quality level. The scaled quantization matrix is then rounded and clipped to have positive

integer values ranging from 1 to 255.

Quantization is achieved by dividing each element in the transformed image matrix' by the corresponding element in the quantization matrix, and then rounding to the nearest integer value. For the following step, quantization matrix Q50 is used.

$$C_{i,j} = round(D_{i,j}/Q_{i,j}) \tag{3.6}$$

Recall that the coefficients situated near the upper-left corner correspond to the lower frequencies - to which the human eye is most sensitive - of the image block. In addition, the zeros represent the less important, higher frequencies that have been discarded, giving rise to the lossy part of compression. As mentioned earlier, only the remaining nonzero coefficients will be used to reconstruct the image. It is also interesting to note the effect of different quantization matrices; use of Q10 would give C significantly more zeros, while Q90 would result in very few zeros.

3.1.6 Coding

The quantized matrix C is now ready for the final step of compression. Before storage, all coefficients of C are converted by an encoder to a stream of binary data (01101011...). In-depth coverage of the coding process is beyond the scope of this article. However, we can point out one key aspect that the reader is sure to appreciate. After quantization, it is quite common for most of the coefficients to equal zero. JPEG takes advantage of this by encoding quantized coefficients in the zig-zag sequence shown in Figure 2.1. The advantage lies in the consolidation of relatively large runs of zeros, which compress very well. The sequence in Figure 1(4x4) continues for the entire 8x8 block.

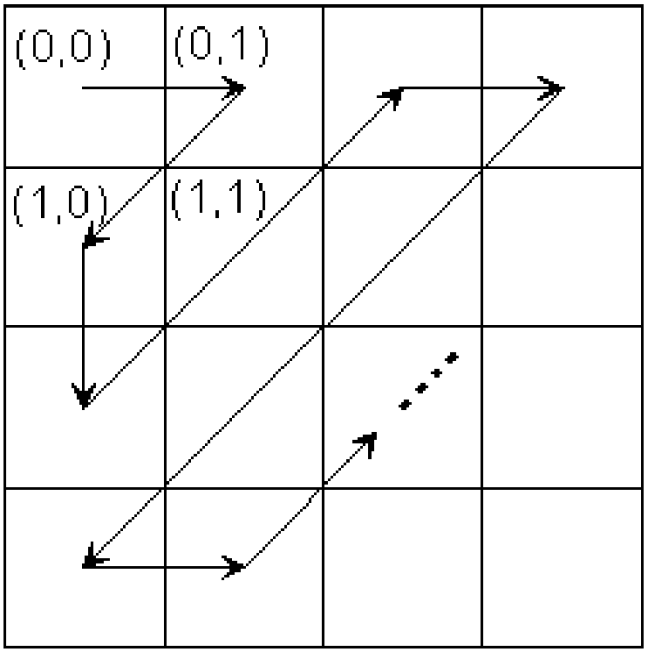

Figure 3.1: Coding in JPEG Algorithm

3.1.7 Decompression

Reconstruction of our image begins by decoding the bit stream representing the quantized matrix C. Each element of C is then multiplied by the corresponding element of the quantization matrix originally used.

$$R_{i,j} = Q_{i,j} * C_{i,j} \tag{3.7}$$

R =

160	44	20	80	24	0	0	0
36	108	14	38	26	0	0	0
-98	-65	16	-48	-40	0	0	0
-42	-85	0	-29	0	0	0	0
-36	22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The IDCT is next applied to matrix 5, which is rounded to the nearest integer. Finally, 128 is added to each element of that result, giving us the decompressed JPEG version 1 of our original 8x8 image block M.

$$N = round(T'^{RT)+128} (3.8)$$

3.2 Embedded Zerotree Wavelet Algorithm

The embedded zerotree wavelet algorithm (EZW) [4] is a simple, yet remarkable effective, image compression algorithm, having the property that the bits in the bit stream are generated in order of importance, yielding a fully embedded code. Using an embedded coding algorithm, an encoder can terminate the encoding at any point thereby allowing a target rate or target distortion metric to be met exactly. Also, given a bit stream, the decoder can cease decoding at any point in the bit stream and still produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated stream. In addition to producing a fully embedded bit stream, EZW consistently produces compression results that are competitive with virtually all known compression algorithms.

3.2.1 Zerotree Data Structure

A wavelet coefficient x is said to be insignificant with respect to a given threshold T if $|x| < T$. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a threshold, then all wavelet coefficients of the same orientation in the same spatial location at the finer scale are likely to be insignificant with respect to the same threshold. More specifically, in a hierarchical subband system, with the exception of the highest frequency subbands, ever coefficient at a given scale can be related to a set of coefficients at the next finer scale of similar orientation. The coefficient at the coarse scale is called the parent, and all coefficients corresponding to the same spatial location at the next finer scale of similar orientation

are called children. Similar, we can define the concepts descendants and ancestors. The data structure of the zerotree can be visualized in Figure 3.2. Given a threshold T to determine whether or not a coefficient is significant, a coefficient x is said to be an element of a zerotree for the threshold T if itself and all of its descendents are insignificant with respect to the threshold T. Therefore, given a threshold, any wavelet coefficient could be represented in one of the four data types: zerotree root (ZRT), isolated zero (IZ) (it is insignificant but its descendant is not), positive significant (POS) and negative significant (NEG)[5].

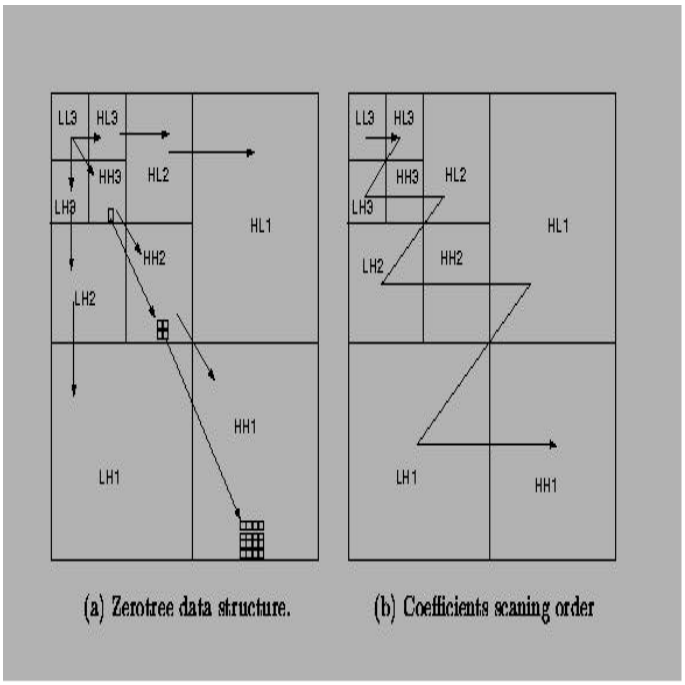

Figure 3.2: Coefficients are coded in a zerotree structure and scanned in a left-to-right order

3.2.2 Dominant Pass

Shapiro's algorithm creates rooted trees using a pixel of the LL subband for the root of each tree and a specific order of similarly positioned pixels from the other subbands for children. There are two types of passes performed: a dominant pass and a subordinate pass. The dominant pass finds pixel values above a certain threshold,

and the subordinate pass quantizes all significant pixel values found in this and all previous dominant passes previous.

A dominant pass checks all trees for significant pixel values with respect to a certain threshold. The initial threshold is chosen to be one-half of the maximum magnitude of all pixel values. Subsequent dominant pass thresholds are always one-half the previous pass threshold. When an insignificant pixel value is found, and a check of all it's children reveals that they too are insignificant, then it is possible to encode that pixel and all it's children with one symbol, a zerotree root, in place of a symbol for that pixel and a symbol for each of that pixel's children, thus achieving compression. Pixel values found to be significant in the dominant pass are encoded with the symbol positive, for a value greater than zero, or negative, for a value less than zero, then those pixel values are added to a subordinate list for quantization, and the pixel value in the subband is then set to zero for the next dominant pass. Pixel values found to be insignificant in the dominant pass but with significant children are coded as isolated zeros. So, the dominant passes map pixel values to a four symbol alphabet which can then be further encoded by using an adaptive arithmetic coder.

3.2.3 Subordinate Pass

After each dominant pass, a subordinate pass is then performed on the subordinate list which contains all pixel values previously found to be significant. The subordinate pass performs pixel value quantization which achieves compression by telling the decoder with a symbol roughly what the pixel value is instead of exactly what the pixel value is. Since the initial threshold is one-half the maximum magnitude of all pixel values for the first dominant pass, then in the first subordinate pass only two ranges are specified in which a significant pixel value could lie: the upper half of the range between the maximum pixel value and the initial threshold, or the lower half of the same range. A pixel value in the upper half of the range gets coded with the symbol upper (for upper part of the range), while a pixel value in the lower half

gets coded with the symbol lower. A pixel value found to be in a particular range is quantized, from the decoders viewpoint, to the midpoint of that range. Upon subsequent subordinate passes the threshold has been cut in half and so there are twice as many ranges as the last subordinate pass plus two new ranges corresponding to the new lower threshold. By reading the subordinate symbol corresponding to a significant pixel and knowing the threshold, the decoder is able to determine the range in which the pixel lies and reconstructs the pixel value to the midpoint of that range. Thus from the decoders viewpoint the rough estimate of a significant pixel's value is getting more refined and accurate as more subordinate passes are made. So, the subordinate passes quantize pixel values to a two symbol alphabet which then get encoded by using an adaptive arithmetic coder as described by Witten, Neal, and Cleary, thus achieving compression.

3.2.4 Working

A very direct approach is to simply transmit the values of the coefficients in decreasing order, but this is not very efficient. This way a lot of bits are spend on the coefficient values and we do not use the fact that we know that the coefficients are in decreasing order. A better approach is to use a threshold and only signal to the decoder if the values are larger or smaller than the threshold. If we also transmit the threshold to the decoder, it can reconstruct already quite a lot. To arrive at a perfect reconstruction we repeat the process after lowering the threshold, until the threshold has become smaller than the smallest coefficient we wanted to transmit. We can make this process much more efficient by subtracting the threshold from the values that were larger than the threshold. This results in a bit stream with increasing accuracy and which can be perfectly reconstructed by the decoder. If we use a predetermined sequence of thresholds then we do not have to transmit them to the decoder and thus save us some bandwidth. If the predetermined sequence is a sequence of powers of two it is called bitplane coding since the thresholds in this case correspond to the bits in the

binary representation of the coefficients. EZW encoding as described in [Sha93] uses this type of coefficient value encoding. One important thing is however still missing: the transmission of the coefficient positions. Indeed, without this information the decoder will not be able to reconstruct the encoded signal (although it can perfectly reconstruct the transmitted bit stream). It is in the encoding of the positions where the efficient encoders are separated from the inefficient ones. As mentioned before, EZW encoding uses a predefined scan order to encode the position of the wavelet coefficients (see figure 2.3). Through the use of zerotrees many positions are encoded implicitly. Several scan orders are possible (see figure 3), as long as the lower subbands are completely scanned before going on to the higher subbands. In [Sha93] a raster scan order is used, while in [Alg95] some other scan orders are mentioned. The scan order seems to be of some influence of the final compression result.

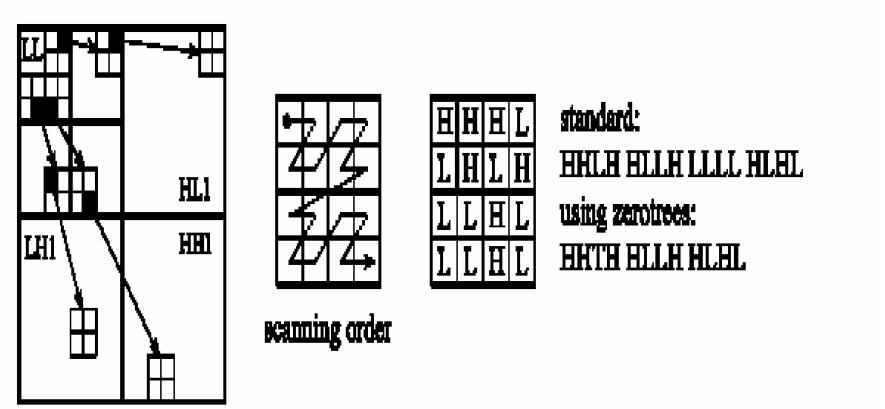

Figure 3.3: The Relation between Wavelet Coefficients in different subbands(left), how to scan them (upper right) and the result of using zerotree(lower right) symbols(T) in the coding process

There are two types of scanning procedures namely

- Raster Scan
- Mortan Scan

The difference in approach between the two procedures is illustrated by the following diagram.

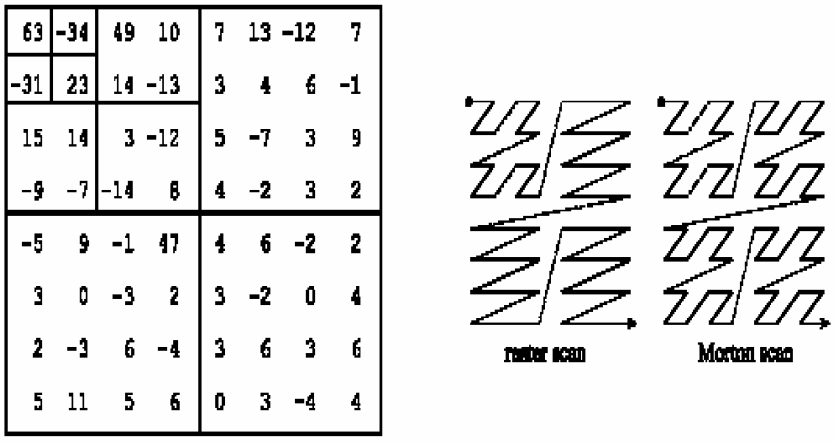

Figure 3.4: The Relation between Wavelet Coefficients in different subbands(left), how to scan them (upper right) and the result of using zerotree(lower right) symbols(T) in the coding process

3.2.5 Algorithm

- Image is read from the file and converted to gray scale.
- Wavelet transform is applied to the gray scaled image according to the size of image using $Decompositionorder = log2(size(originalimage))$
- Generate wavelet coefficients
- Convert wavelet coefficients to matrix format
- Encode wavelet coefficients using EZW algorithm

Encoding stops when final threshold value is achieved, the procedure is discussed in below diagram

Huffman coding algorithm is applied to the EZW bit stream to achieve the final compressed image.

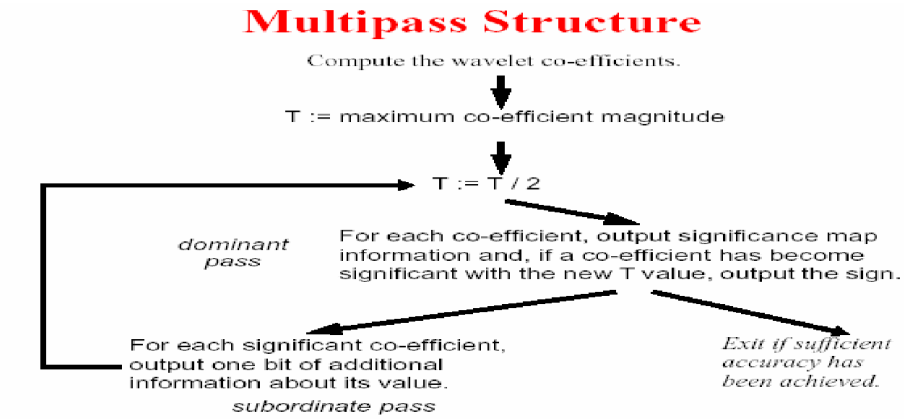

Figure 3.5: Algorithm

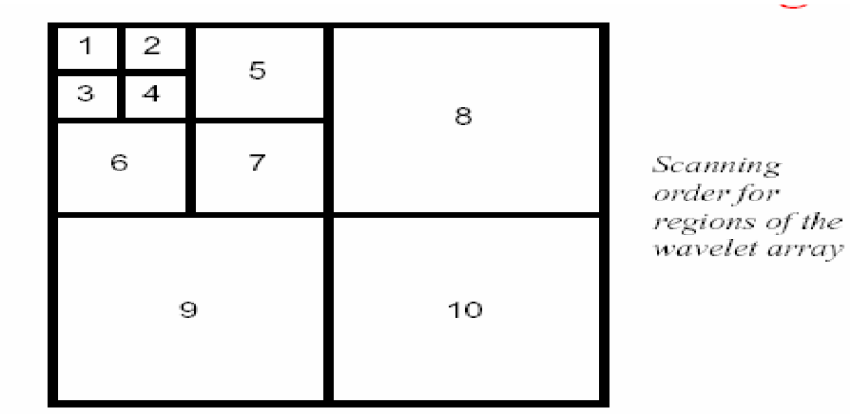

Figure 3.6: Wavelet Co-efficient scanning order

Chapter 4

Implementation & Results

4.1 EZW Implementation with Matlab

The Embedded Zerotree Wavelet (EZW) algorithm is implemented with the Matlab and the Wavelet Toolbox.The procedure of the implementation is as following.

- reading the gray scale images (256 X 256)
- defining wavelet dcomposition order
  $Decompositionorder = \log(size(originalimage))$
- Discrete wavelet transform to generate wavelet co-efficient
- EZW Encoding
- Huffman Encoding
- Huffman Decoding
- EZW Decoding using the same threshold in Encoding
- Inverse DWT
- Writing the Reconstructed Image

4.2 Image Compression Comparision Parameters

Following are the well known parameters for the image compression algorithm comparision.

- Compression Ratio
- PSNR
- Execution Time
- Size of compressed Image
- Bit per Pixel (bpp)

4.2.1 PSNR Ratio

- The PSNR is most commonly used as a measure of quality of reconstruction of lossy image compression.
- The signal in this case is the original data, and the noise is the error introduced by compression.
- It is most easily defined via themean squared error(MSE)

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} ||I(i,j) - K(i,j)||^2 \tag{4.1}$$

$$PSNR = 10 * log_{10}(\frac{MAX_I^2}{MSE}) = 20 * log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{4.2}$$

4.3 Result

4.3.1 Lena Image

- Threshold = 25

  Bitrate = 0.71 bpp PSNR = 27.65

- Threshold = 50

  Bitrate = 0.31 bpp PSNR = 26.21

4.3.2 Cameraman Image

- Threshold = 25

  Bitrate = 0.71 bpp PSNR = 27.65

- Threshold = 50

  Bitrate = 0.31 bpp PSNR = 26.21

4.3.3 Peppers Image

- Threshold = 25

  Bitrate = 0.71 bpp PSNR = 27.65

- Threshold = 50

  Bitrate = 0.31 bpp PSNR = 26.21

Figure 4.1: Original Lena Image

4.4 Result Analysis

As from the result it is clear that for the lower initial threshold at the encoding the bitrate of the reconstructed image is larger compare to the larger initial threshold at the encoding. i.e. for the lena image if the initial threshold = 25 then the bitrate = 0.71 bpp and for initial threshold = 50 the bitrate = 0.31 bpp. From the original images and the reconstructed images we can see that for the lower initial threshold at encoding the reconstructed image is more clear compare to the higher initial threshold at the encoding in the EZW Algorithm.

Figure 4.2: Reconstructed Lena Image with Threshold = 25

Figure 4.3: Reconstructed Lena Image with Threshold = 50

Figure 4.4: Original Cameraman Image

Figure 4.5: Reconstructed Cameraman Image with Threshold = 25

Figure 4.6: Reconstructed Cameraman Image with Threshold = 50

Figure 4.7: Original Peppers Image

Figure 4.8: Reconstructed Peppers Image with Threshold = 25

Figure 4.9: Reconstructed Peppers Image with Threshold = 50

Chapter 5

Conclusion & Future Work

5.1 Conclusion

This work has proposed the wavelet based image comrpression technique. Wavelet is the new technology in the image processing and image compression. Using Discrete Wavelet Transform (DWT) in the image compression gives better result compare to the Discrete Fourier Transform(DFT). In this thesis the Embedded Zerotree Wavelet(EZW) algorithm is implemented. The EZW is the efficient algorithm among the wavelet image compression algorithms.

5.2 Future Scope

The future scope of the thesis goes in the direction of the better image compression using the wavelets. The JPEG2000 standard is the next generation image compression standard which includes the wavelet computations. So using the concepts of the EZW algorithm and wavelets the implementation of the JPEG2000 is possible.

References

[1] T. Instruments, "Tms320c6000 technical brief, literature no : Spru197d,"

[2] T. Instruments, "Tms320c6000 technical peripherals, literature no : Spru1900,"

[3] K. Cabeen and P. Gent, "Image compression and the discrete cosine transform,"

[4] P. D. S. N. T. V.S.SHINGATE, Dr. T.R.Sontakke, "Still image compression using embedded zerotree wavelet encoding,"

[5] J. M. Shapiro, "Smart compression using the embedded zerotree wavelet (ezw) algorithm,"